Clemens Teichmann

Spezifikation, Implementierung und Evaluierung einer browserbasierten Benutzeroberfläche für das Warenwirtschaftssystem onrooby

DIPLOMARBEIT

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCE

Informationstechnik & Elektrotechnik

Mittweida, 2009

Clemens Teichmann

Spezifikation, Implementierung und Evaluierung einer browserbasierten Benutzeroberfläche für das Warenwirtschaftssystem onrooby

eingereicht als

DIPLOMARBEIT

an der

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCE

Informationstechnik & Elektrotechnik

Potsdam, 2009

Erstprüfer: Prof. Dr. rer. nat. Sergej Alekseev

Zweitprüfer: Dipl.-Wirt.-Inf. Lars Thielemann

Vorgelegte Arbeit wurde verteidigt am:

Teichmann, Clemens:
Spezifikation, Implementierung und Evaluierung einer browserbasierten Benutzeroberfläche für das Warenwirtschaftssystem onrooby
Hochschule Mittweida (FH), Fachbereich Informationstechnik & Elektrotechnik
Diplomarbeit, 2009

Referat:
Die onrooby GmbH entwickelt das Warenwirtschaftssystem onrooby für den Online-Handel. Für die Benutzeroberfläche existiert bereits ein Konzept, welches in Teilen schon umgesetzt wurde. Diese Arbeit beschreibt das Konzept und erläutert den Entwicklungsstand. Des Weiteren wird, auf Basis des Konzepts, die Benutzeroberfläche spezifiziert. Im Anschluss wird die spezifizierte Benutzeroberfläche implementiert. Darüber hinaus werden Methoden entwickelt, zur kritischen Bewertung der Benutzeroberfläche.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Ajax	Asynchronous JavaScript and XML
B1	Bearbeitungsebene 1
B2	Bearbeitungsebene 2
CSS	Cascading Style Sheets.
DTD	Dokumenttyp-Definition
DOM	Document Object Model
GF	Gewichtungsfaktor
GIF	Graphics Interchange Format
HTML	Hypertext Markup Language
IE	Internet Explorer
JPEG	Joint Photographic Experts Group
NW	Nutzwert
NWA	Nutzwertanalyse
Rails	Ruby on Rails
PNG	Portable Network Graphics
PRUG	Process routed user guidance
WWS	Warenwirtschaftssysteme
WWW	World Wide Web
(X)HTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
ZF	Zielerfüllungsfaktor

1 Einleitung

1.1 Motivation

Immer mehr Menschen haben Zugang zum Internet und nutzen das *World Wide Web* (*WWW*). Das *WWW* hat sich in den letzten Jahren rasant weiterentwickelt: Statische Seiten wurden von dynamischen, interaktiven Webanwendungen verdrängt. Mit Hilfe der dynamischen Webanwendungen kann das *WWW* als Oberfläche für verteilte Programme genutzt werden. Das hat den Vorteil, dass Programme nicht mehr lokal installiert werden müssen, sondern dezentral (auf einem Server) administriert werden. Dadurch werden Ressourcen gespart und das Programm kann von überall aus bedient werden.
Die onrooby GmbH entwickelt mit onrooby ein innovatives, browserbasiertes Warenwirtschaftssystem. Ich bekomme die Möglichkeit, eine moderne Benutzeroberfläche zu implementieren und eigene Ideen mit einzubringen. Ich werde an der Herausforderung wachsen und ein junges, modernes IT-Unternehmen kennenlernen.

1.2 Kapitelübersicht

Im **Kapitel 2** geht es zunächst um die Software onrooby und dessen Softwaretypus.
Des Weiteren wird das vorhandene Konzept für die Benutzeroberfläche beschrieben und geklärt, inwieweit es schon umgesetzt wurde.

Kapitel 3 erläutert grundlegende Technologien und Verfahren für die Bearbeitung des Themas.

Das **Kapitel 4** spezifiziert das Aussehen, die Funktionen und Besonderheiten der Benutzeroberfläche.

Kapitel 5 beschreibt die Entscheidungsfindung für eine geeignete JavaScript-Bibliothek.

Im **Kapitel 6** wird erläutert, wie die Oberfläche technisch umgesetzt wurde.

Das **Kapitel 7** stellt Maßnahmen zur Evaluierung der Benutzeroberfläche vor.

In **Kapitel 8** werden die Ergebnisse der Arbeit zusammengefasst und ein Ausblick gegeben.

Anhang A beinhaltet die Nutzwertanalyse zur Beurteilung der Eingabeelemente.

Anhang B enthält die Messdaten des *Slickspeed*-Geschwindigkeitstest.

Anhang C beinhaltet die Quelltexte der relevanten (X)HTML-Dokumente.

Anhang D enthält die Quelltexte der relevanten Ruby-Funktionen.

Im **Anhang E** sind allgemeine JavaScript- und jQuery-Quelltexte zu finden.

Anhang F enthält die JavaScript- und jQuery-Quelltexte für das Ziehen und Loslassen von Cards.

Anhang G enthält die CSS-Formatierungen.

Anhang H beinhaltet einen Fragebogen zur Benutzeroberfläche von onrooby.

2 Analyse und Abgrenzung der Aufgabenstellung

2.1 Warenwirtschaftssysteme

2.1.1 Allgemeines

Warenwirtschaftssysteme (WWS) sind computergestützte Informationssysteme, zur Steuerung und Abbildung von Geschäftsprozessen; [1] S.3 sie dienen dazu wiederkehrende Aufgaben zu automatisieren und zu vereinfachen. Ein weiterer Vorteil ist das verteilte Wissen einer solchen Anwendung; die Mitarbeiter können schnell und problemlos die Abläufe und Zusammenhänge des Unternehmens einsehen und steuern.

Ein WWS kann beliebig viele Bereiche eines Unternehmens abdecken; die drei Kernbereiche sind der Einkauf, Verkauf und die Lagerhaltung.
Im Einkauf werden automatisierte Bestellungen ausgelöst; eine Bestellung wird bei Bedarf ausgelöst und ist vom Lagerbestand und Verkaufsdaten abhängig.
Der Verkauf erstellt mit Hilfe des WWS Angebote, Aufträge und Rechnungen.
Die Lagerhaltung überwacht die interne Bestandsführung. Durch den Kauf, Verkauf und Verbrauch von Gütern ergeben sich ständig Veränderungen im Bestand.

Nach Auffassung von Hertel lässt sich ein WWS in vier Ebenen aufteilen:
Die erste Ebene ist das Warenprozessmodell; es repräsentiert die physischen Warenvorgänge wie Entladen, Einlagern und Transportieren.
Die zweite Ebene ist das Dispositionsmodell; dispositive Prozesse betreffen nicht die Waren selber, aber werden von diesen ausgelöst: Zum Beispiel Warenbestellungen, Auftragseingang und Rechnungseingang.
Bei der dritten Ebene handelt es sich um das Abrechnungsmodell; es wiederspiegelt die Erfassung von Einkauf- und Verkaufszahlen, sowie Konditionen. Die Zahlen und Konditionen ergeben sich aus dem Zusammenspiel der ersten beiden Ebenen.
Die vierte Ebene sammelt die Informationen der ersten drei Ebenen; sie dient der Steuerung und Optimierung von Preisen, Beständen und Sortiment. [1] S. 4-6

2.1.2 Onrooby

Onrooby ist ein neues Warenwirtschaftssystem und wird voraussichtlich im Januar 2010 fertiggestellt. Die onrooby GmbH entwickelt das WWS für mittelständige Unternehmen; zurzeit sind 4 Programmierer und ein Grafiker mit der Fertigstellung beschäftigt. Das WWS wird mit dem quelloffenen Webentwicklungs-Framework Rails (Ruby on Rails) programmiert; als Oberfläche wird ein Webinterface entwickelt. Das WWS kann auf unterschiedliche Weise betrieben werden:
Zum einen kann das WWS im Betrieb installiert werden. Die Mitarbeiter benutzen das WWS lokal und über das betriebseigene Netzwerk – es wird keine Internetverbindung benötigt.
Darüber hinaus kann ein Kunde, dass WWS als Dienstleistung über das Internet nutzen. Dabei wird die Anwendung von einem professionellen Rechenzentrum gehostet. Die dezentrale Nutzung hat den Vorteil, dass die Installation und Wartung schneller und einfacher ist. Dadurch entstehen für den Kunden weniger Kosten.
Das WWS beinhaltet eine Bestandsführung, ein Kundenbeziehungsmanagement, eine Aufgabenliste, Wirtschaftsprognose und einen Verkaufs- und Bestellprozess.
Darüber hinaus können eigene Geschäftsprozesse angelegt und konfiguriert werden. In einem Geschäftsprozess werden Daten abstrahiert dargestellt und manipuliert.
Dafür verfügt das WWS über eine Reihe von Datenmodellen, welche die relevanten Bestandteile einer Wirtschaft repräsentieren. Ein Datenmodell ist zum Beispiel eine Aktivität, Adresse, Adressentyp, Artikel und Bestellung; diese können voneinander abhängig sein. Die Datenmodelle können in einem gewissen Rahmen vom Betrieb erweitert werden; es ist beispielsweise möglich neue Adresstypen anzulegen.
Das WWS bietet verschiedene Schnittstellen zum Onlinehandel; über die Schnittstellen lassen sich Waren, aus dem WWS heraus, auf beliebten Verkaufsportalen (ebay, Amazon) automatisiert vertreiben. Dafür ist es notwendig, einmalig festzulegen, wie die Ware auf dem Portal angezeigt wird. Danach kann die Ware per Knopfdruck angeboten werden. Sobald jemand den Artikel kauft, werden die relevanten Verkaufsdaten an das WWS übermittelt. Wenn das WWS den Geldeingang verzeichnet, werden die weiteren Prozesse zur Abwicklung des Geschäfts ausgelöst; es wird zum Beispiel der Warenbestand aktualisiert, eine Rechnung erstellt und der Versand vorbereitet.

2.2 Konzept für die Benutzeroberfläche von onrooby

2.1.1 Grundgedanke

Das Programm sollte möglichst einfach und intuitiv zu bedienen sein; daraus ergibt sich ein geringer Schulungsaufwand. Der Endkundenbereich soll ohne externe Hilfe zu bedienen sein – die Onlinehilfe umfasst alle Instruktionen.

2.1.2 Cards

Cards sind Elemente zum Präsentieren und Manipulieren von Daten; sie werden auf der Oberfläche platziert und können unterschiedliche Höhen und Breiten haben. Eine komplexe Anordnung von Cards dient dem Abhandeln einer bestimmten Aufgabe (Prozess); deshalb wird eine solche Anordnung auch Prozess genannt. Idealerweise muss der Benutzer zum Abhandeln eines Prozesses nur eine Seite aufrufen; d.h. alle relevanten Daten sind auf der Oberfläche verfügbar.
In einem Prozess können Cards voneinander abhängig sein; wenn man in einer Card Daten sucht, anzeigt oder verändert, wird der Inhalt aller abhängigen Cards ggf. aktualisiert.

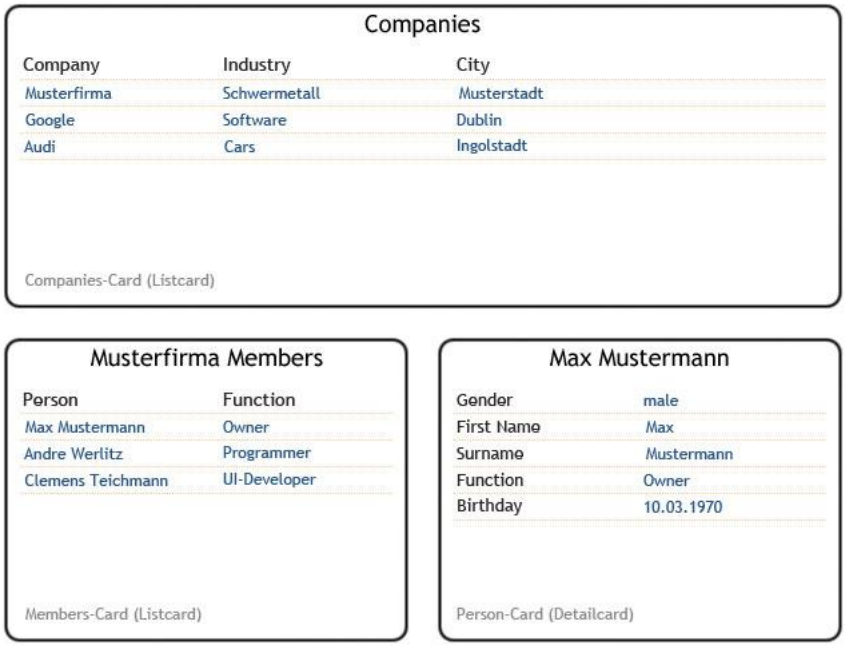


Abbildung 2-1: Anordnung von Cards (Prozess)

In Abbildung 2-1 sind beispielsweise die beiden unteren Cards voneinander abhängig: Wenn man in der Members-Card einen Mitarbeiter auswählt, werden die entsprechenden Daten in der Person-Card aktualisiert.

Dabei gibt es zwei verschiedene Arten von Cards:

(1) In einer **Listcard** werden mehrere Datensätze in einer Liste angezeigt; dabei entspricht jede Zeile einem Datensatz. In Abbildung 2-1 repräsentieren die Companies- und Members-Card eine Listcard

(2) Der zweite Card-Typ ist die **Detailcard**: In der Detailcard werden Daten eines Datensatzes angezeigt und bearbeitet. In Abbildung 2-1 repräsentiert die Person-Card eine Detailcard.

2.1.3 Process routed user guidance (PRUG)

Jeder Prozess dient dem Abhandeln einer bestimmten Aufgabe. Dabei bekommen alle beteiligten Cards nacheinander den Fokus zur Bearbeitung; die Reihenfolge in der die Cards den Fokus bekommen, kann variieren und ist vom Benutzer abhängig. Typischerweise gibt es eine optimale oder schnellste Reihenfolge.
PRUG ist ein Konzept, dass den Benutzer bei dem Abhandeln eines Prozesses unterstützt und optimierte Reihenfolgen vorschlagen kann.

Im Detail existieren folgende Ansätze:

- die Card mit dem Fokus soll hervorgehoben werden
- Cards die sich durch Benutzeraktionen verändern, werden hervorgehoben
- sobald eine Card den Fokus bekommt, wird das erste Feld in der Card selektiert
- nach dem man eine Card bearbeitet hat, kann eine festgelegte Nachfolger-Card automatisch den Fokus erhalten

2.3 Ist-Zustand

2.3.1 Allgemeines

Die onrooby GmbH hat den Entwicklungsprozess des WWS in Versionsschritte aufgeteilt; es gibt die Versionen 0.1 bis 1.0 mit der Schrittgröße 1/10; es gibt also 10 verschiedene Versionen und die Version 1.0 ist die finale Version.
In jeder Version werden neue Features erstellt, die das System erweitern und verbessern. Eine neue Version wird in einem Zeitraum von 4 Wochen entwickelt.
Zu Beginn meiner Arbeiten war der Entwicklungsstand des WWS in der Version 0.3; In dieser Version waren grundlegende Funktionen implementiert; man konnte neue Prozesse anlegen und konfigurieren, sowie Daten erstellen, manipulieren und löschen.

2.3.2 Oberfläche

Die Oberfläche setzt sich aus HTML (*Hypertext Markup Language*), CSS (*Cascading Style Sheets*) und Grafiken zusammen. Die Oberfläche der Version 0.3 wurde für die Darstellung im Firefox-Browser[1] entwickelt und soll von den Mitarbeitern zu Entwicklungs- und Testzwecken genutzt werden.
Darüber hinausgehende Anforderungen wie Browser-Kompatibilität, valides HTML und CSS wurden nicht beachtet.
So kommt es zum Beispiel im IE[2] (Internet Explorer) zu Darstellungsfehlern, eine HTML-Validierung ergibt 26 Fehler und eine CSS-Validierung 10 Fehler.
Die grundlegende Aufteilung des Raumes wurde wie folgt umgesetzt: Am oberen Rand findet man alle Prozesse nebeneinander gelistet; darunter werden die Cards in Reihen und Spalten platziert; unter den Cards befindet sich eine Fußleiste.
Die Aufteilung des Raumes soll auch in künftigen Versionen der Oberfläche so bleiben.
Das Design der Oberfläche ist sehr einfach gestaltet. Es enthält drei unterschiedliche Farben und graue Buttons

[1] http://www.mozilla-europe.org/de/firefox/
[2] http://www.microsoft.com/germany/windows/internet-explorer/

2.3.3 Cards

Das Card-Konzept wurde in Version 0.3 in Teilen umgesetzt:
Die Prozesse werden in der Datenbanktabelle “business_processes” und die Cards in der Datenbanktabelle “cards” gespeichert. Die Tabelle “business_processes” hat eine 1:n-Assoziation mit der “cards”-Tabelle (siehe Abbildung 2-2); das bedeutet, dass ein Prozess in Beziehung zu beliebig vielen Cards stehen kann und jede Card zu genau einem Prozess gehört.

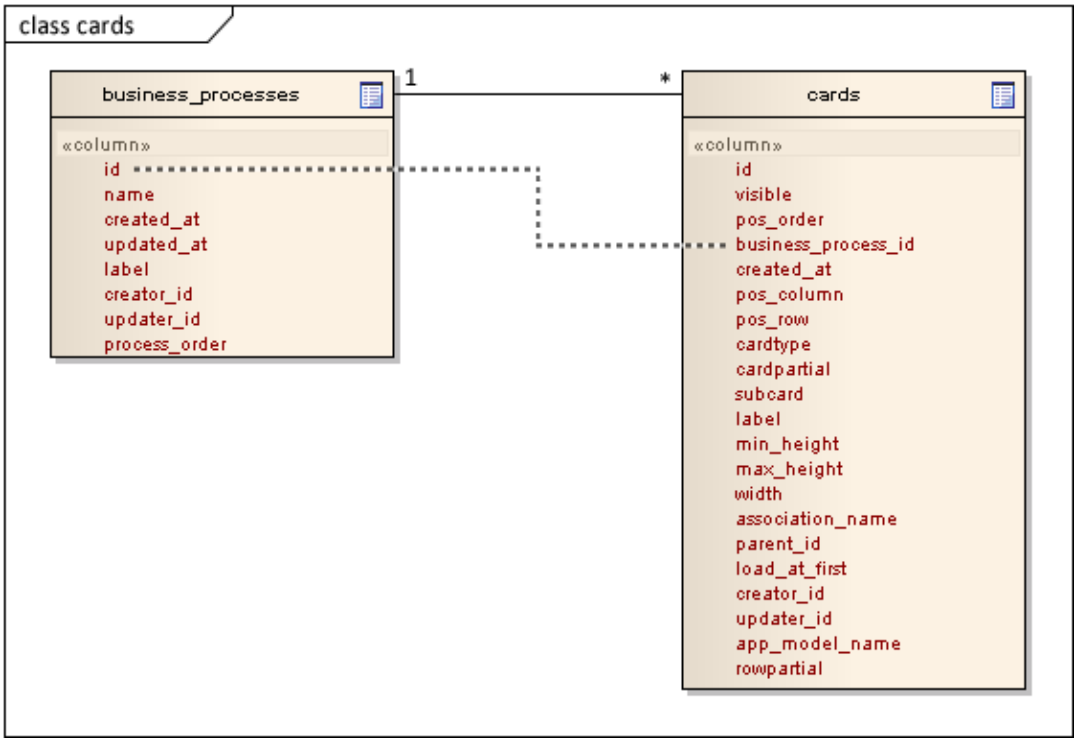


Abbildung 2-2 Datenbank-Tabellen

Zur Ausgabe wird ein Raster mit fünf Reihen und fünf Spalten generiert; bei der Generierung wird für jede Position (Reihe, Spalte) geprüft, ob dort eine Karte gespeichert ist; falls ja, wird die Karte an dieser Position ausgeben; falls nicht, wird ein Platzhalter an dieser Stelle ausgegeben. Dieser Algorithmus ist in Abbildung 2-3 dargestellt:

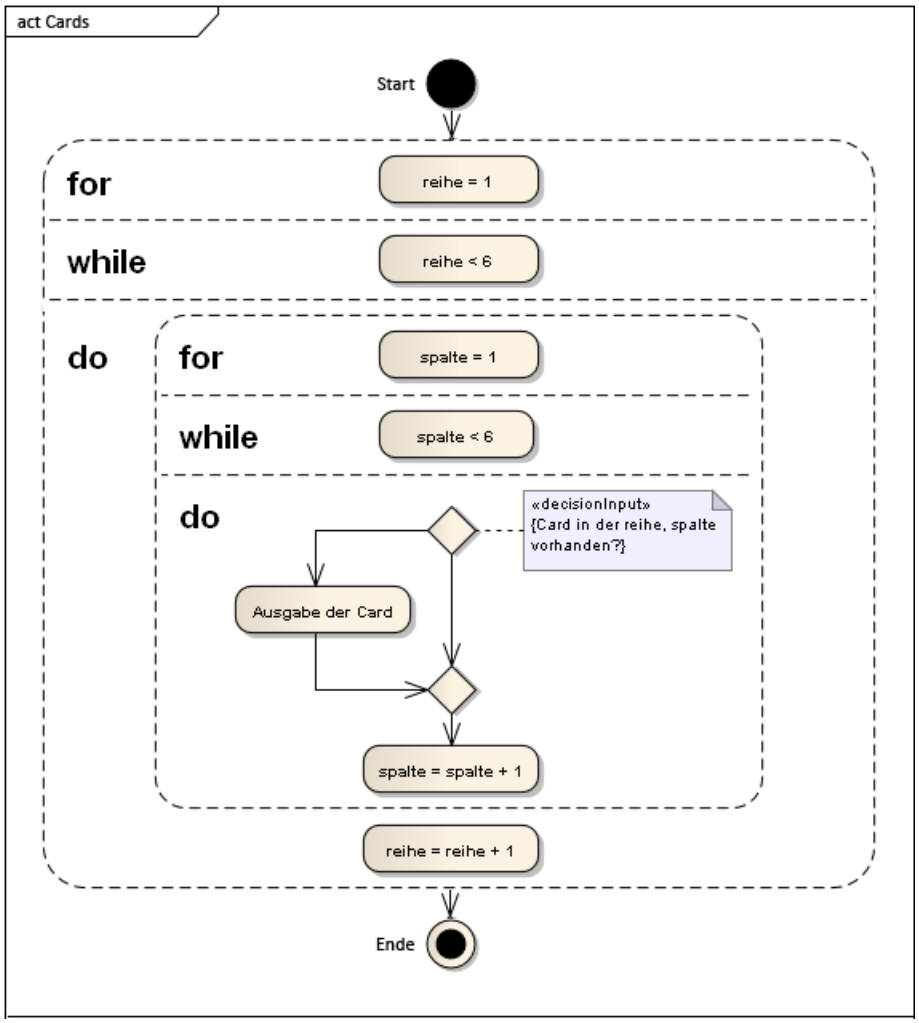


Abbildung 2-3 Aktivitätsdiagramm Cards

Prozesse und Cards lassen sich in mit Hilfe des “Prozess”-Prozesses erstellen, bearbeiten und löschen. Die Eigenschaften einer Card können in einem Formular eingestellt werden. (Abbildung 2-4) Dazu gehören unter anderem die Position (“pos row”, “pos column”), die Breite (“width”) und Höhe (“height”). Die Breite repräsentiert dabei die Anzahl der Spalten und die Höhe wird in Pixeln angegeben.

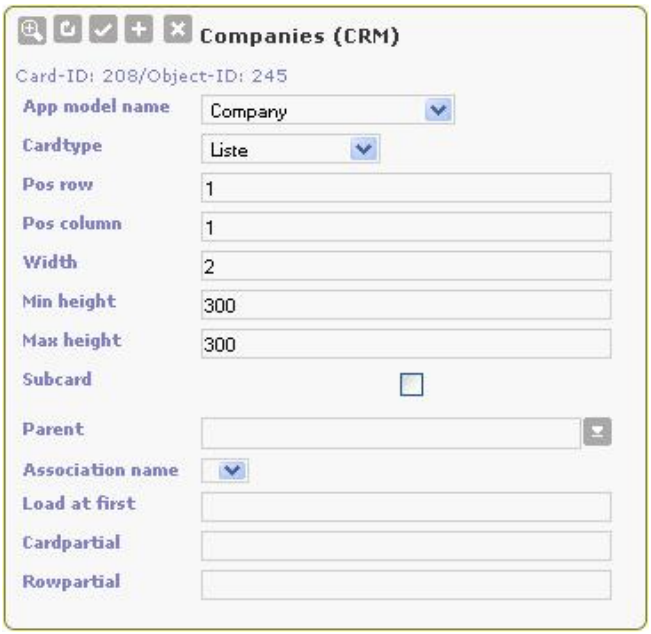


Abbildung 2-4: Cards-Konfiguration

2.3.4 PRUG

Folgende Aspekte von PRUG wurden bereits umgesetzt:

- Wenn man Daten in einer Card bearbeitet und speichert, wird die Card mit einem visuellen Effekt kurz hervorgehoben.
- Während man Daten in einer Card bearbeitet, werden die veränderten Daten mit einem roten Rahmen markiert.
- Cards mit Subcards können verschiedene Aktionen auslösen, die das Neuladen der Daten in der Subcard erfordern; dabei wird die Subcard mit einem visuellen Effekt für kurze Zeit hervorgehoben. Eine Subcard ist eine Card, die von einer anderen Card abhängig ist.

3 Grundlagen

3.1 Ruby on Rails

3.1.1 Einführung

Rails (Ruby on Rails) ist ein Framework für die Entwicklung von Webanwendungen; es wurde von David Heinemeier Hansson mit der Programmiersprache Ruby entwickelt und 2004 quelloffen veröffentlicht. Das Framework entstand durch die Arbeit an dem Projektmanagementprogramm basecamp[1]. [2] Kap. 1.1
2006 gelang Rails ein erster Durchbruch – damals gab Apple bekannt, dass Rails mit Mac Os 10.5 Leopard ausgeliefert wird. [3] S. 153 Inzwischen setzen auch große Unternehmen auf Rails – zum Beispiel Amazon, eBay und XING.
Das Framework wird beständig weiterentwickelt; die Weiterentwicklung wird vom *Rails Core Team*[2] gesteuert und vorangetrieben. An der Weiterentwicklung des Frameworks sind inzwischen mehr als 1400 Menschen beteiligt. [4] Rails ist aktuell in der Version 2.3.4 verfügbar. Die dritte Version wurde bereits angekündigt; in Version 3 wird Rails mit dem zweiten großen Ruby-Webframework merb zusammen geführt und soll die besten Aspekte aus beiden übernehmen. [5]

Rails folgt den Prinzipien *don't repeat yourself* und *convention over configuration*.
Don't repeat yourself bedeutet Redundanzen zu vermeiden: Jeder Programmcode sollte möglichst einmalig vorkommen. Dadurch lässt sich der Code einfacher warten und Inkonsistenten werden vermieden. [3] S. 157
Convention over configuration meint, dass sich der Entwickler nur um ungewöhnliche Konfigurationen kümmern muss. Für die meisten Aspekte eines Webframeworks hat Rails eine Standardkonfiguration – dadurch spart sich der Entwickler den Konfigurationsaufwand. Trotzdem besteht die Möglichkeit, die Standardkonfigurationen zu überschreiben. [2] Kap. 1.5

[1] http://basecamphq.com/
[2] http://rubyonrails.org/core

3.1.2 Die Programmiersprache Ruby

Ruby ist eine vielseitige höhere Programmiersprache, die von Yukihiro Matsumoto entwickelt wurde. Ruby ist durchgängig objektorientiert – hat aber auch prozedurale, funktionale und imperative Paradigmen; darüber hinaus bietet Ruby dynamische Typisierung und automatische Speicherverwaltung. Die Standardimplementierung ist in der Programmiersprache C geschrieben. Ruby ist eine interpretierte Sprache: Programme werden zur Laufzeit eingelesen, analysiert und ausgewertet.
Ruby erschien zum ersten mal 1995 und ist frei zugänglich[1]. Der Name Ruby leitet sich von dem gleichnamigen Edelstein *ruby* (Rubin) ab. [3] S. 3

Yukihiro Matsumoto hatte die Motivation eine Sprache zu entwickeln, die stärker als Perl und objektorientierter als Python ist. Ruby wurde dementsprechend von den beiden Sprachen beeinflusst; weitere Einflüsse kamen von den Sprachen Lisp und Smalltalk. [6]
Der Japaner legte Wert darauf, wie sich die Sprache anfühlt; sie sollte leicht verständlich sein und der des Menschen ähneln: [7]

```
# folgende Zeile Code gibt drei mal 'Lorem ipsum' aus
3.times puts "Lorem ipsum"
```

3.1.3 Architektur

Rails strukturiert eine Anwendung nach dem Architekturmuster “Model View Controller“ – das impliziert die Trennung von *model* (Datenmodell), *view* (Präsentationsschicht) und *controller* (Programmsteuerung).

In Rails werden ***models*** mit ActiveRecord erstellt. ActiveRecord ist ein Framework, dass den objektorientierten Zugriff auf relationale Datenbanksysteme steuert;
es verbindet Datenbanktabellen mit Objekten, in denen die Logik und die Daten vereint sind; ActiveRecord ist nach dem *active record pattern*[2] von Martin Fowler aufgebaut. ActiveRecord beinhaltet Methoden zum Suchen, Bearbeiten, Speichern und Löschen von Daten; darüber hinaus enthält ein *model* Attribute, die sich auf die Spalten der Da-

[1] http://www.ruby-lang.org/en/LICENSE.txt
[2] http://martinfowler.com/eaaCatalog/activeRecord.html

tenbanktabelle beziehen. Um die Firma mit der *id* 1 zu laden und dessen Namen zu ändern, würde man in Rails folgenden Code schreiben:

```ruby
# finde die Company mit der id 1
company = Company.find_by_id(1)
# ändere den Namen der Company in 'foo'
company.name = "foo"
# speicher die Änderungen
company.save
```

Des Weiteren lassen sich mit ActiveRecord Assoziation zwischen den *models* knüpfen. Solche Assoziationen werden über Schlüsselwörter wie “belongs_to”, “has_many” und “has_one” definiert. Eine Assoziation repräsentiert häufig eine Relation zwischen zwei Datenbanktabellen.

Für die Präsentation der Daten (***view***) ist in Rails das Modul ActionView verantwortlich; es definiert drei unterschiedliche *templates*:
Dateien die auf “html.erb” enden, sind eine Mischung aus Ruby und HTML. Ruby-Code wird auf folgende Weise in HTML eingebettet:

```erb
<h1>
// innerhalb <% %> wird Rubycode ausgewertet
<%foo = "bar"%>
// innerhalb <%=  %> wird Rubycode ausgewertet
// und als String ausgegeben
<%=foo%>
</h1>
```

Das zweite *template* endet auf “js.rjs” – diese Dateien enthalten Ruby-JavaScript; damit lässt sich mit Ruby JavaScript erzeugen. Soll eine “Hallo Welt”-Meldung nach einer Sekunde ausgegeben werden, würde man mit Ruby-JavaScript folgenden Code schreiben:

```ruby
page.delay(1) do
  page.alert("Hallo Welt")
end
```

Templates die auf .xml.builder enden, werden zur Präsentation von XML (Extensible Markup Language) verwendet.

Controller sind Klassen, welche die Logik der Anwendung ausführen; sie besitzen öffentliche Methoden, die ihre *actions* repräsentieren.
Rails beinhaltet ein vorkonfiguriertes *routing*, dass eine Anfrage an einen *controller* und eine zugehörige *action* weiterleitet. Zum Beispiel würde der Aufruf von "http://localhost/company/new" ein Objekt des *controllers* "Company" erzeugen und die Methode "new" aufrufen.

3.1.4 Namenskonventionen

Namen von Variablen und Dateien werden in Rails klein geschrieben; wenn der Name aus mehreren Wörtern besteht, werden diese mit einem Unterstrich verbunden.
Namen von Datenbanktabellen haben dieselbe Konvention wie Variablen und Dateinamen; darüber hinaus werden die Namen von Datenbanktabellen in der Mehrzahl geschrieben.
Die Namen der *controller*, *models* und Module werden groß geschrieben; jedes neue Wort fängt mit einem Großbuchstaben an. Des Weiteren werden alle *models* in der Einzahl geschrieben.

3.2 JavaScript und das Document Object Model

JavaScript ist eine Skriptsprache, mit dessen Hilfe sich Web-Seiten optimieren lassen. [8] Die Sprache wird hauptsächlich für DOM-Manipulationen in Web-Browsern eingesetzt. JavaScripts werden zur Laufzeit client-seitig vom Browser interpretiert.
Das DOM (Document Object Model) definiert, wie Elemente des Dokuments referenziert und manipuliert werden können. Es handelt sich dabei um ein allgemeines Model, welches vom *W3-Consortium*[1] herausgegeben wird. [8]
Um mit dem DOM zu arbeiten, wird zunächst das Dokument eingelesen und ein Dokumenten-Objekt erzeugt. Auf Basis dieses Objekts kann dann der Inhalt, die Struktur und Darstellung des Dokuments mit JavaScript gelesen und verändert werden.

3.3 Asynchronous JavaScript and XML

Ajax (*Asynchronous JavaScript and XML*) ist ein Konzept zur asynchronen Datenübertragung zwischen dem Browser und einem Server; es ermöglicht eine HTTP-Anfrage durchzuführen, ohne die HTML-Seite komplett neu zu laden.

[1] http://www.w3.org/

Der Begriff Ajax bezeichnet keine eigene Technologie – sondern den Technologiemix, der Ajax ausmacht. [9]

Für eine Webanwendung mit Ajax-Funktionalität benötigt man eine HTML-Seite, JavaScript und das XMLHttpRequest-Objekt.

XMLHttpRequest ist eine Programmierschnittstelle, zum Transfer von beliebigen Daten über das HTTP-Protokoll; dabei wird auf das erneute Laden der gesamten HTML-Seite verzichtet. Das XMLHttpRequest-Objekt wird mit Hilfe von JavaScript erzeugt und gesteuert. Die Daten, die als Antwort vom Server über das XMLHttpRequest-Objekt geschickt werden, können dynamisch in die Seite integriert werden; dafür manipuliert JavaScript die Seite mit Hilfe des DOM.

Herkömmliche Webanwendungen werden bei jeder Benutzeraktion komplett neugeladen: der Benutzer sendet einen *request* an den Server, der mit der angeforderten HTML-Seite antwortet (Abbildung 3-1); dabei kann sich die Antwort des Servers verzögern, wodurch lange Wartezeiten und sogar Brüche auftreten können.

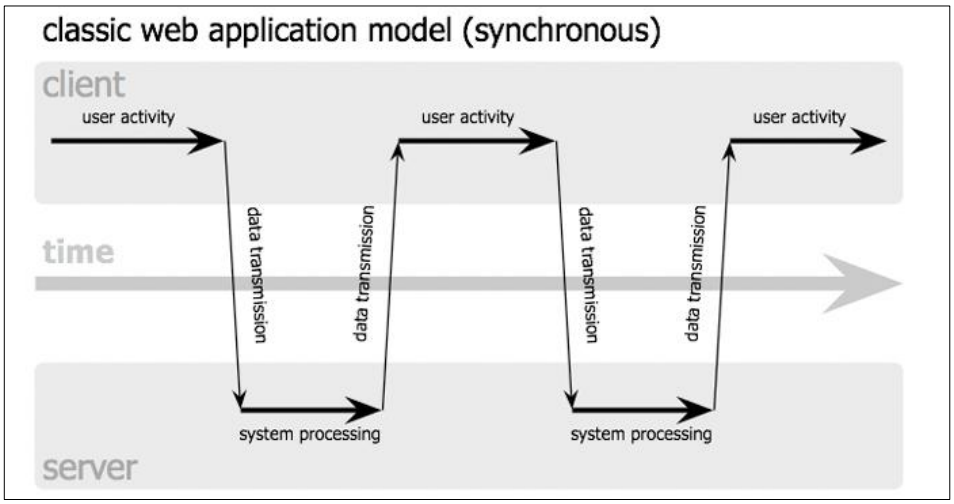


Abbildung 3-1: classic web application model [9]

Bei Ajax-Anwendungen muss der Benutzer nicht auf die Antwort des Servers warten; er kann die Webanwendung weiter nutzen, während der Browser mit dem Server kommuniziert. (Abbildung 3-2) Die angeforderten Ressourcen werden bei Bedarf in die Seite integriert. Das verkürzt die Wartezeiten und verhindert, dass überflüssige Daten mit übertragen werden;

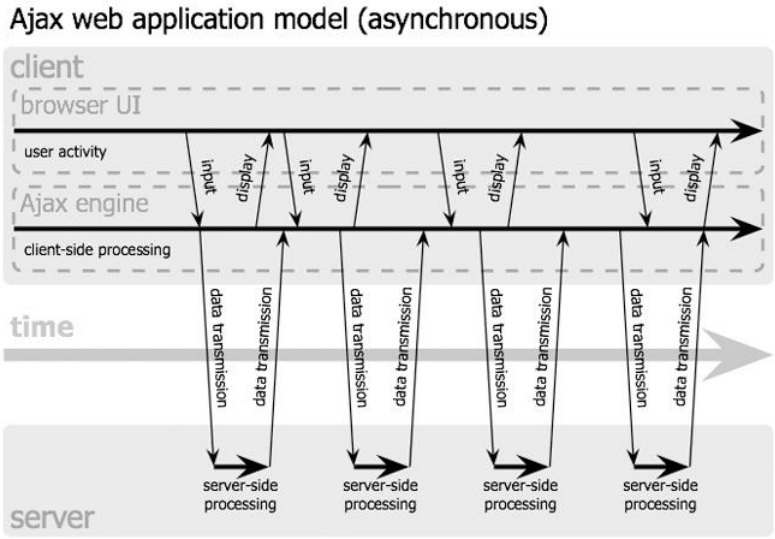


Abbildung 3-2: Ajax web application model [9]

Ein wichtiges Erfolgskriterium der onrooby-Applikation ist ein flüssiger Arbeitsfluss; es soll der Eindruck einer desktopähnlichen Anwendung entstehen. Um das zu gewährleisten, wird die Anwendung größtenteils über Ajax mit dem Server kommunizieren.

3.4 Nutzwertanalyse

3.4.1 Allgemeines

Die NWA (Nutzwertanalyse) ist eine Methode zur Entscheidungsfindung, die den Nutzwert zu verschiedenen Entscheidungsalternativen liefert; dabei liefert die beste Alternative den höchsten Nutzwert.

Die NWA soll im Rahmen dieser Arbeit dazu verwendet werden, eine geeignete JavaScript-Bibliothek zu finden. Es gibt noch eine Reihe anderer Methoden zur Entscheidungsfindung; ich habe mich für die NWA entschieden, da diese prägnant und transparent ist; die NWA bietet einen neuen Blickwinkel auf die Entscheidungskriterien und vergleicht die Alternativen mit nachvollziehbarer Methodik. [10]

3.4.2 Entscheidungsalternativen und Bewertungskriterien

Der erste Schritt der NWA ist, alle Entscheidungsalternativen aufzuschreiben. Im zweiten Schritt werden alle Bewertungskriterien aufgeschrieben, die zur Entscheidung herangezogen werden; dabei sollte man nur die wichtigsten Kriterien berücksichtigen, die schließlich zur Entscheidung führen sollen.

Um die NWA zu veranschaulichen, wird begleitend eine fiktive NWA durchgeführt; in dem Beispiel gibt es die Entscheidungsalternativen Netzteil A, Netzteil B, Netzteil C und die Bewertungskriterien Preis, Leistung und Qualität.

3.4.3 Gewichtungsfaktoren

Als nächstes werden die festgelegten Kriterien nach Gewichtung miteinander verglichen; dafür werden alle Kriterien in einer Tabelle gegenübergestellt; dies wird in Tabelle 1 beispielhaft gemacht:

Kriterium	Preis	Leistung	Qualität
Preis			
Leistung			
Qualität			

Tabelle 3-1: Kriterien der fiktiven NWA

Jetzt beginnt man oberhalb der Diagonalen, für jede Zeile die Kriterien miteinander zu vergleichen; dabei stellt man sich die Frage, ob das Kriterium aus dem Reihenkopf wichtiger als das Kriterium aus dem Spaltenkopf ist: nein ergibt den Punktwert 0, genauso wichtig 1 und wichtiger 2; den Punktwert trägt man in die entsprechende Zelle ein.
In dem Beispiel ist der Preis wichtiger als die Leistung und wichtiger als die Qualität, die Leistung ist genauso wichtig wie die Qualität; dies führt zu Tabelle 3-2:

Kriterium	Preis	Leistung	Qualität
Preis		2	2
Leistung	0		1
Qualität	0	1	

Tabelle 3-2: Prioritäten der fiktiven NWA

Dabei reicht es, sich die Frage für die Punktwerte oberhalb der Diagonalen zu stellen. Diese Punktwerte werden dann an der Diagonalen gespiegelt und negiert eingetragen.

Als nächstes wird die Summe jeder Zeile, in eine neue Spalte “Gewicht” eingetragen; addiert man alle Gewichtungswerte erhält man die Gewichtsumme – die wird für die Ermittlung der Gewichtsfaktoren gebraucht. Der Gewichtsfaktor ist der Gewichtungswert, geteilt durch die Gewichtssumme; daraus ergibt sich Tabelle 3-3:

Kriterium	Preis	Leistung	Qualität	**Gewicht**	**Gewichtsfaktor**
Preis		2	2	4	**0,66**
Leistung	0		1	1	**0,16**
Qualität	0	1		1	**0,16**
			Gewichtssumme:	6	

Tabelle 3-3: Gewichtungsfaktoren der fiktiven NWA

Die Gewichtsfaktoren spiegeln den Anteil wieder, den die Kriterien an der Entscheidung haben; sie werden später zur Berechnung des tatsächlichen Nutzwertes herangezogen.

3.4.4 Skala der Zielerfüllungsfaktoren

Im nächsten Schritt wird eine Skala aufgestellt, in der festgehalten wird, wie die einzelnen Kriterien erfüllt werden können; man spricht in diesem Zusammenhang von den Zielerfüllungsfaktoren. Für die Zielerfüllungsfaktoren wählt man sich einen beliebigen Zahlenbereich, wobei die kleinste Zahl das Kriterium gar nicht erfüllt und die höchste Zahl das Kriterium voll erfüllt. In der Skala werden die Zielerfüllungsfaktoren durch Bereiche aufgeteilt, die verbal beschrieben werden. In diesem Beispiel könnte eine solche Skala so aussehen:

Skala / **Kriterium**	0-2	3-5	6-8
	schlecht	mittel	gut
Preis	über 100 €	50 - 100 €	unter 50 €
Leistung	unter 500 W	500 - 600 W	über 600 W
Qualität	schlecht verarbeitet; hoher Verschleiß	solide verarbeitet; mäßiger Verschleiß	gut verarbeitet; geringer Verschleiß

Tabelle 3-4: Zielerfüllungsfaktoren der fiktiven NWA

Diese Skala (Tabelle 3-4) veranschaulicht für jedes Kriterium, die Bedeutung der Zielerfassungsfaktoren.

3.4.5 Nutzwerte und Ergebnis

Im letzten Schritt werden schließlich die Nutzwerte ermittelt. Dazu schreibt man zunächst, einen Zielerfüllungsfaktor für jede Alternative und jedes Kriterium in eine Tabelle; in dem Beispiel könnte das wie folgt aussehen:

Alternative / Kriterium	Netzteil A	Netzteil B	Netzteil C
Preis	7	5	1
Leistung	2	5	8
Qualität	2	4	8

Tabelle 3-5: Eingetragene Zielerfüllungsfaktoren der fiktiven NWA

Diese Tabelle kombiniert man jetzt mit dem Gewichtungsfaktor; dabei wird für jede Alternative mit jedem Kriterium ein Nutzwert ausgerechnet, wobei sich der Nutzwert aus der Multiplikation des Zielerfüllungs- und des Gewichtsfaktor ergibt.
Aus der Addition der einzelnen Nutzwerte errechnet sich schließlich ein Index, für den Nutzen der einzelnen Alternativen. In diesem Beispiel ergibt sich folgende Tabelle:

		Netzteil A		**Netzteil B**		**Netzteil C**	
	Gewichtungsf.	Zielerfüllungsf.	Nutzwert	Zielerfüllungsf.	Nutzwert	Zielerfüllungsf.	Nutzwert
Preis	0,66	7	4,62	5	3,3	1	0,66
Leistung	0,16	2	0,32	5	0,8	8	1,28
Qualität	0,16	2	0,32	4	0,64	8	1,28
		Index:	**5,26**	Index:	**4,74**	Index:	**3,22**

Tabelle 3-1: Nutzwerte der fiktiven NWA

Die Alternative mit dem höchsten Index hat dabei den meisten Nutzen; in dem Beispiel hat Netzteil A am meisten Nutzen und wäre somit die beste Wahl.

4 Spezifikation der Oberfläche

4.1 Definition

Die Oberfläche wird in drei Bereiche (Abbildung 4-1) unterteilt:

Abbildung 4-1: Aufteilung der Oberfläche

Die Kopfleiste beinhaltet das Firmenlogo und die verschiedenen Prozesse; sie wächst dynamisch nach rechts, so dass alle Prozesse in die Leiste passen.
Die Cards eines Prozesses werden im Prozesskörper platziert; dieser breitet sich mit wachsendem Inhalt dynamisch nach rechts und unten aus.
Die Fußleiste enthält zusätzliche Informationen zum Programm (Versionsnummer, Copyright).

4.2 Design

Das Design (Abbildung 4-2) wurde von einem Grafiker angefertigt und liegt als Photoshop-Datei vor; es soll möglichst detailgetreu umgesetzt werden.

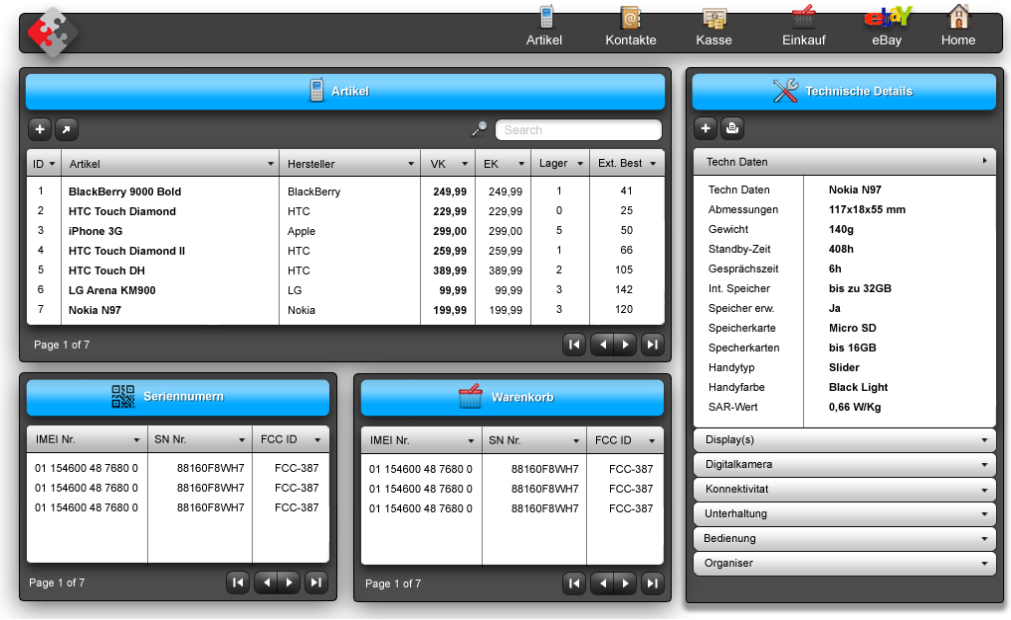


Abbildung 4-2: Design der Benutzerfläche

4.3 Cards

4.3.1 Definition

Cards sind abgegrenzte Bereiche auf der Oberfläche, in denen Daten dargestellt und bearbeitet werden. Es können beliebig viele Cards in einem Prozess definiert werden; dabei sollen sich einzelne Cards nie überlappen. Jede Card hat eine Breite, Höhe, Reihen- und Spaltenposition. Jede Card besteht aus einer Titelleiste, einem Inhaltsbereich und einer Fußleiste:

Abbildung 4-3: Aufbau einer Card

Wenn der Inhalt höher ist, als der dafür vorgesehene Bereich, wächst der Bereich automatisch nach unten. Dabei wird die Fußzeile entsprechend nach unten verschoben und die Card wächst nach unten.

4.3.2 Positionierung durch Ziehen und Loslassen

Der Benutzer soll eine Card mit der Maus an eine bestimmte Position verschieben können. Dabei soll es auch möglich sein, die Card an eine bereits belegte Position zu verschieben. In diesem Fall sollen die Cards die Positionen tauschen.

4.3.3 Größenänderung mit der Maus

Die Card-Breite und Höhe soll sich durch Mausaktionen verändern lassen. Diese Möglichkeit soll ein Benutzer wahrnehmen können, sobald der Mauszeiger den Card-Rand erreicht hat: Der Mauszeiger tauscht sein Icon mit einer entsprechenden Grafik – zum Beispiel zwei vertikalen Pfeile bzw. zwei horizontalen Pfeilen. Jetzt kann der Benutzer den Rand der Card, mit der linken Maustaste fixieren und bei gedrückter Taste verschieben. Die Card kann nur soweit vergrößert werden, wie freier Platz vorhanden ist; wird ein Platz durch eine andere Card besetzt, muss diese zunächst verkleinert oder verschoben werden. Darüber hinaus soll eine Mindestbreite und Höhe nie unterschritten werden.

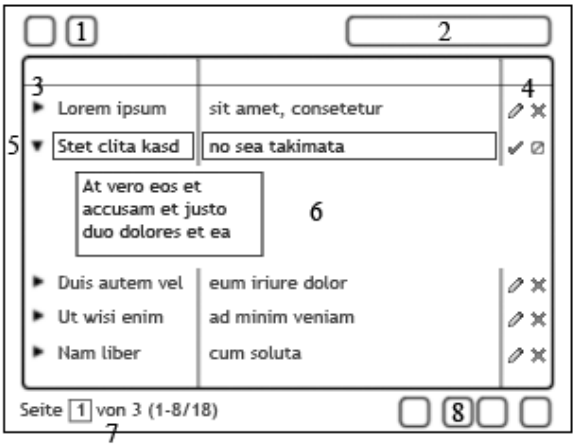
4.3.4 Inkonsistente Cards

Ungesicherte Daten in einer Card sollen kenntlich gemacht werden; ungesicherte Daten entstehen, wenn man einen neuen Datensatz anlegt oder einen vorhandenen bearbeitet hat. Die Daten sind solange ungesichert, bis sie gespeichert werden.
Um dem Benutzer vor Augen zu führen, dass eine Card ungesicherte Daten besitzt, soll diese Card sich vom normalen Card-Design unterscheiden.
Des Weiteren sollen die bearbeiteten Formular-Felder hervorgehoben werden. Wenn der Benutzer die Daten speichert, sollen die Card und die Formular-Felder wieder das normale Aussehen annehmen.
Bevor die Seite verlassen wird, soll geprüft werden, ob ungesicherte Daten vorhanden sind; falls ja wird eine entsprechende Warnung ausgegeben und der Vorgang kann abgebrochen werden.

4.4 Listcard

4.4.1 Definition

Die Listcard ist ein bestimmter Cardtyp, zum Darstellen und Bearbeiten von Datensätzen in einer Liste. Der Inhalt einer Listcard ist in Abbildung 4-4 veranschaulicht:



1 - Aktion-Buttons
2 - Suchfeld
3 - Pfeil zum öffnen und schließen der zweiten Bearbeitungsebene
4 Icon zum öffnen der Bearbeitungsebene 1; Icon zum löschen des Datensatzes
5 Bearbeitungsebene 1
6 Bearbeitungsebene 2
7 Seiteninformation mit Eingabefeld
8 Buttons zum Seitenblättern

Abbildung 4-4: Inhalt einer Listcard

In einer Listcard sollen Daten angelegt, bearbeitet, gelöscht, gesucht und gelesen werden; dafür gibt es verschiedene Elemente und Mechanismen:

Die Liste enthält eine beliebige Menge von Datensätzen, wobei jede Zeile einen Datensatz repräsentiert. Es werden immer soviel Datensätze angezeigt, wie in die Card passen. Nicht angezeigte Datensätze können durch blättern (Abbildung 4-4 / 8) oder Eingabe einer Seitenzahl (Abbildung 4-4 / 7) erreicht werden. Über ein Suchfeld (Abbildung 4-4 / 2) lassen sich Daten in der Liste suchen; dabei werden in der Liste nur noch die Datensätze angezeigt, die der Suche entsprechen.

Am Ende jeder Zeile befindet sich ein Icon zum Löschen des Datensatzes und zum Öffnen der Bearbeitungsebene 1. (Abbildung 4-4 / 4)
In der Bearbeitungsebene 1 werden Daten in einer Zeile bearbeitet: Die gesamte Zeile wird zu einem Formular, welches den Datensatz repräsentiert. Die einzelnen Formularelemente sollen genau an der Stelle erscheinen, wo woher die Daten standen; sie werden also innerhalb der Spalten platziert. Für die unterschiedlichen Formular-Elemente (Textfeld, Textbox, Selectbox) gibt es eine Mindestbreite. Wenn die Spaltenbreite die Mindestbreite unterschreitet, wird das Formular-Element entsprechend abgeschnitten. So-

bald das Formular-Element bearbeitet wird, soll es vollständig angezeigt werden; dann soll es sich über die anderen Inhalte legen. Beim Öffnen der Bearbeitungsebene werden die Icons am Zeilenende ausgetauscht: Es gibt jetzt ein Icon zum Speichern und eins zum Abbrechen.

Die Bearbeitungsebene 1 soll bei jeder Card-spezifischen Aktion automatisch gespeichert und geschlossen werden. Dadurch wird verhindert, dass Daten versehentlich verloren gehen. Des Weiteren kann so immer nur eine Bearbeitungsebene gleichzeitig geöffnet sein.

Eine weitere Bearbeitungsebene wird durch ein Pfeil-Icon geöffnet und wieder geschlossen. (Abbildung 4-4 / 3) Diese zweite Bearbeitungsebene ist zum Bearbeiten von Datenfeldern, die nicht Bestandteil der Zeile sind. Die Datenfelder der Bearbeitungsebene 2 sollen vom Benutzer eingestellt werden; die Position eines Datenfeldes innerhalb der Ebene, wird durch den Abstand nach oben und links definiert – der Bezugspunkt ist die linke obere Ecke der zweiten Bearbeitungsebene. Darüber hinaus lässt sich die Höhe der Bearbeitungsebene einstellen – diese wird in Anzahl von Zeilen angegeben. Wird eine zweite Bearbeitungsebene geöffnet, sollen entsprechend der Höhe, Zeilen der Liste ausgeblendet werden; es sollen die Zeilen ausgeblendet werden, die am weitesten entfernt in der angezeigten Liste vom bearbeitenden Datensatz liegen.
Zum Beispiel öffnet ein Benutzer eine zweite Bearbeitungsebene in der Zeile 5; die Bearbeitungsebene hat die Höhe 2; es werden insgesamt 10 Datensätze in der Liste angezeigt: Dann soll die Zeile 1 und 10 ausgeblendet werden.
Beim Schließen der Ebene erscheinen die vorher ausgeblendeten Zeilen wieder. Der Anwender soll durch einen flüssigen visuellen Effekt wahrnehmen, wenn eine zweite Bearbeitungsebene oder Zeilen ein- und ausgeblendet werden. Die erste Bearbeitungsebene wird automatisch geöffnet, wenn eine zweite Ebene geöffnet wird.

Mit Hilfe der Aktion-Buttons (Abbildung 4-4 / 1) lassen sich neue Datensätze anlegen und vorhandene kopieren. Wenn ein neuer Datensatz angelegt wird, wird eine neue Zeile am Kopf der Liste generiert. Die Zeile erscheint als Bearbeitungsebene 1, damit sofort Daten eingegeben werden können. Sobald der Datensatz gespeichert wird, soll die Liste neu geladen werden. Wenn der Datensatz verworfen wird, soll die Zeile wieder verschwinden.

4.4.2 Fokussieren einer Zeile

Wenn eine Zeile den Fokus bekommt, soll sich diese von den anderen Zeilen abheben. Eine Zeile bekommt den Fokus, wenn der Benutzer den Mauszeiger über der Zeile führt oder mit Hilfe von Tastenkombinationen die Zeile auswählt.
Die Zeile verliert den Fokus, sobald der Mauszeiger die Zeile verlässt oder der Benutzer mit Tastenkombinationen in eine andere Zeile navigiert.

4.4.3 Tastaturbedienung

Eine Listcard soll mit Hilfe der Pfeiltasten bedient werden können; das folgende Zustandsdiagramm (Abbildung 4-5) soll den Ablauf verdeutlichen:

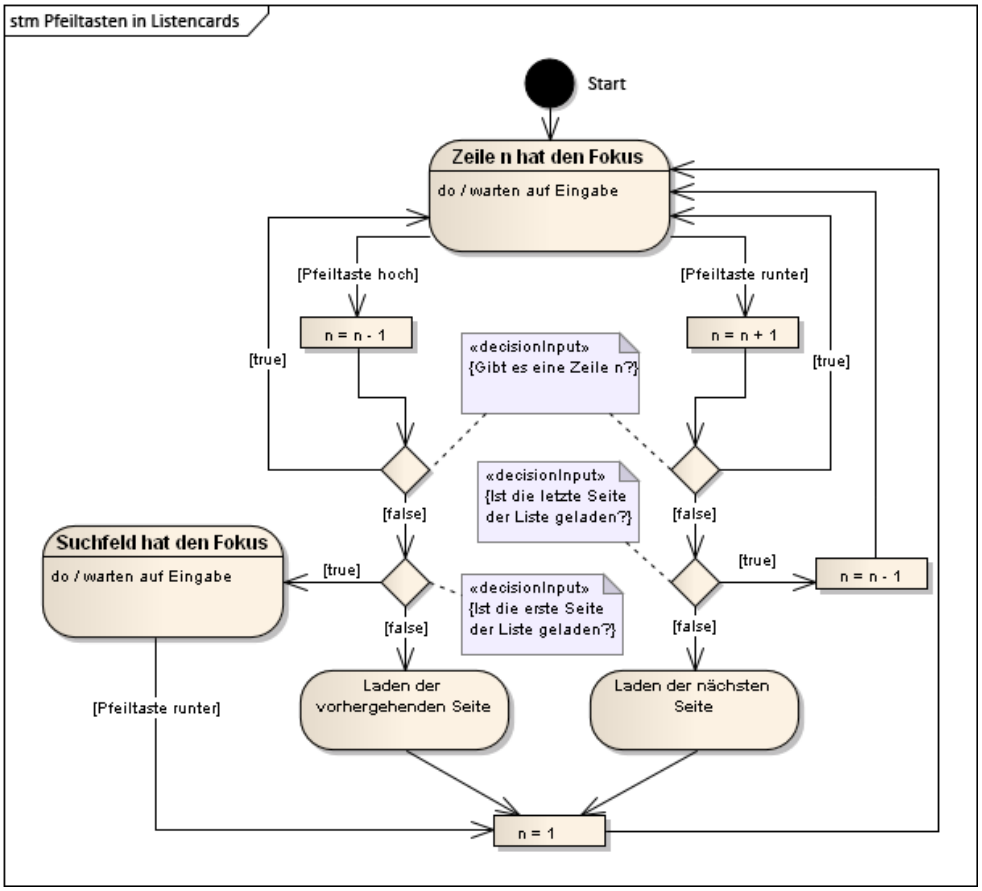


Abbildung 4-5: Zustandsdiagramm Pfeiltasten in einer Listcard

4.5 Detailcard

Ein weiterer Card-Typ ist die Detailcard. In einer Detailcard wird ein Datensatz in einem Formular dargestellt. Die Formularfelder werden auf der rechten Seite untereinander gelistet; die dazu gehörigen Feldbeschriftungen befinden sich auf der linken Seite. Genau wie in einer Listcard, besitzt eine Detailcard in der linken oberen Ecke Aktion-Buttons; mit deren Hilfe lassen sich neue Datensätze anlegen, kopieren, speichern und löschen.

4.6 PRUG

4.6.1 Beziehungen zwischen Cards

Beim Abhandeln eines Prozesses werden verschiedene Cards nacheinander bearbeitet. Dabei haben manche Cards typische Nachfolgercards in der Abarbeitungsreihenfolge. Wenn der Benutzer eine Card fertig bearbeitet hat und für diese Card eine Nachfolgercard definiert ist, dann soll die Nachfolgecard hervorgehoben werden. Die Hervorhebung soll in Form eines leichten visuellen Effektes erfolgen.

4.6.2 Beziehungen zwischen Zeilen und Cards

Bestimmte Datensätze (Zeilen) verlinken mit ihnen assoziierte Aktionen: Abhängige Daten werden in anderen Cards neu geladen. Die Beziehung zwischen Datensatz und Card soll visualisiert werden; dafür soll die Zeile dauerhaft und die neugeladene Card kurzzeitig hervorgehoben werden. Die dauerhafte Markierung soll dem Benutzer vor Augen führen, welcher Datensatz gerade aktiv ist. Diese Markierung soll solange aktiv bleiben, bis in der Card ein anderer Datensatz ausgewählt wurde.

4.6.3 Workflow-Recorder

Der Workflow-Recorder ist ein Tool, mit dem der Benutzer Bearbeitungsabfolgen von Cards aufzeichnen und verwalten kann.
Eine Aufzeichnung soll durch einen Aufnahme-Button gestartet werden; dann bearbeitet ein Benutzer verschiedene Cards – dabei merkt sich der Workflow-Recorder die Reihenfolge. Die Aufnahme wird durch erneutes Klicken des Aufnahme-Buttons beendet. Der Benutzer hat jetzt die Möglichkeit, die Aufnahme unter einem gewünschten Namen zu speichern. Die Aufzeichnung kann später wieder geladen werden.

Wenn eine Aufzeichnung geladen ist, soll sich die Geschwindigkeit und Orientierung für den Anwender beim Abarbeiten eines Prozesses verbessern:
Sobald eine Card fertig bearbeitet wurde, soll die Nachfolgercard hervorgehoben werden (siehe Abschnitt 4.6.1). Darüber hinaus soll - falls vorhanden - das erste Formularfeld der Nachfolgercard den Fokus bekommen; dadurch können Daten sofort eingegeben werden, ohne vorher das Feld mit der Maus anzuwählen.

4.6.4 Aktiven Prozess hervorheben

In der Kopfleiste befinden sich Links auf die Prozesse, wobei immer nur ein Prozess gleichzeitig aktiv sein kann. Der aktive Prozess soll sich in der Kopfleiste von den anderen abheben; dadurch soll der Benutzer stets vor Augen haben, welcher Prozess gerade aktiv ist.

5 Auswahl einer JavaScript-Bibliothek

5.1 Einleitung

JavaScript ist ein wichtiges Werkzeug für die Implementierung der Oberfläche; es wird für Ajax-Aufrufe und dynamisches HTML benötigt. Dynamisches HTML bezeichnet Webseiten, die nach dem Aufruf der Seite veränderbar bleiben. Es können zum Beispiel Teile ausgetauscht oder animiert werden.
Eine JavaScript-Bibliothek soll den Entwicklungsaufwand verringern, wobei es mehrere unterschiedliche Bibliotheken gibt. Die Entscheidung für eine bestimmte Bibliothek ist sorgfältig zu treffen, sodass sie auch zukünftigen Anforderungen gerecht wird. Denn ein späterer Wechsel der Bibliothek ist mühselig: Der gesamte Code muss für die neue Bibliothek angepasst werden und die Entwickler müssen sich in eine neue Code-Syntax einarbeiten.
Die Entscheidungsfindung sollte alle Aspekte berücksichtigen und die verschiedenen Alternativen nachvollziehbar vergleichen; um das zu gewährleisten, soll im Folgenden eine NWA (Nutzwertanalyse) zur Auswahl einer JavaScript-Bibliothek durchgeführt werden.

5.2 Entscheidungsalternativen

Die Entscheidungsalternativen wurden durch die onrooby GmbH vorgegeben, wobei alle Bibliotheken folgende Mindestanforderungen erfüllen:

- DOM-Manipulation
- Ajax-Funktionalitäten

Es sollen die Bibliotheken Dojo Toolkit, Ext Js, jQuery und Prototype untersucht werden. Die unterschiedlichen Bibliotheken werden wie folgt vorgestellt:

Dojo Toolkit

"Ajax, events, packaging, CSS-based querying, animations, JSON, language utilities, Skinnable, template-driven widgets with accessibility and localization built right in the way you want it. From accordions to tabs, we have you covered. Inventive & innovative code and widgets. Visualize your data with grids and charts. Take your apps offline. Cross-browser vector drawing. And a lot more."[1]

[1] http://www.dojotoolkit.org/

Ext JS

"Ext JS is a cross-browser JavaScript library for building rich internet applications. It includes:

- *High performance, customizable UI widgets*
- *Well designed and extensible Component model*
- *An intuitive, easy to use API*
- *Commercial and Open Source licenses available"*[1]

jQuery

"jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript."[2]

Prototype

"Prototype is a JavaScript Framework that aims to ease development of dynamic web applications. Featuring a unique, easy-to-use toolkit for class-driven development and the nicest Ajax library around, Prototype is quickly becoming the codebase of choice for web application developers everywhere."[3]

5.3 Bewertungskriterien

Die folgenden Kriterien sollen die Entscheidung beeinflussen:

- **Dateigröße** - Wie klein ist der Kern des Frameworks?
- **Eingabeelemente** - Inwiefern erfüllen die Eingabeelemente die Anforderungen der onrooby GmbH?
- **Geschwindigkeit** - Wie schnell arbeitet die Bibliothek?
- **Hilfsquellen** - Wie ist die Dokumentation der Bibliothek? Wie ist die Unterstützung in der Entwicklergemeinschaft?
- **Kosten -** Wie viel Geld muss aufgebracht werden, um die Bibliothek kommerziell nutzen zu können?

[1] http://www.extjs.com/
[2] http://jquery.com/
[3] http://www.prototypejs.org/

5.4 Gewichtungsfaktoren

Kriterium	Dateigröße	Eingabeelemente	Geschwindigkeit	Hilfsquellen	Kosten	Gewichtung	Gewichtungsfaktor
Dateigröße		0	0	1	0	1	0,05
Eingabeelemente	2		1	2	1	6	0,3
Geschwindigkeit	2	1		2	1	6	0,3
Hilfsquellen	1	0	0		0	1	0,05
Kosten	2	1	1	2		6	0,3
					Summe:	20	

Tabelle 5-1: Gewichtungsfaktoren der NWA

5.5 Skala der Zielerfüllungsfaktoren

(1) Die Skalenwerte der **Dateigröße** reichen von 0 bis 100 KB.

(2) Da es mehrere unterschiedliche Eingabeelemente gibt, wurde eine eigenständige NWA mit den Eingabeelementen als Bewertungskriterien durchgeführt. (siehe Anhang A) Die Nutzwerte variierten dabei zwischen 0,75 und 3. Dementsprechend geht die Skala der **Eingabeelemente** von 0 bis 3.

(3) Um die Hilfsquellen zu bewerten, wird eine Umfrage[1] von Kyle Hayes genutzt. In dieser Umfrage wird unter anderem nach der Qualität der Dokumentation und der Unterstützung der Anwender und Entwickler gefragt. Aus diesen beiden Werten wurde ein Mittelwert gebildet. Die Skala der **Hilfsquellen** reicht von 4 bis 5,5.

(4) Die Geschwindigkeit der einzelnen Bibliotheken, wurde mit dem Slickspeed-Geschwindigkeitstest[2] gemessen. Bei diesem Test wird die Dauer für das Selektieren von DOM-Elementen gemessen. Der Test wurde in verschiedenen Browsern mehrfach durchgeführt. (siehe Anhang B) Aus den Messwerten wurde ein Mittelwert gebildet. Die Skala für die **Geschwindigkeit** reicht von 50 bis 95 ms (Millisekunden).

(5) Die **Kostenskala** geht von 0 bis 750 €.

[1] http://www.kylehayes.info/2009/03/29/survey-results-javascript-frameworks/
[2] http://www.domassistant.com/slickspeed/

Daraus generiert sich folgende Tabelle:

Skala / Kriterium	0-1	2-3	4-5
	mangelhaft	**befriedigend**	**gut**
Dateigröße (KB)	mehr als 100	50 - 100	0 - 50
Eingabeelemente (Nutzwert)	0 - 1	1 - 2	2 - 3
Geschwindigkeit (ms)	80 - 95 ms	65 - 80 ms	50 - 65 ms
Hilfsquellen (Umfragewert)	4 - 4,5	4,5 - 5	5 - 5,5
Kosten (€)	500 – 750 € und mehr	250 – 500 €	0 – 250 €

Tabelle 5-2: Zielerfüllungsfaktoren der NWA

5.6 Nutzwerte und Ergebnis

In der folgenden Tabelle sind die Werte eingetragen, die zur Ermittlung der Zielerfüllungsfaktoren gebraucht werden:

Framework / Kriterium	Dojo Toolkit	Ext Js	jQuery	Prototype
Dateigröße (KB)	26	84	19	46
Eingabeelemente (Nutzwert)	2	3	0,25	0,75
Geschwindigkeit (ms)	65,33	64,83	51	89,83
Hilfsquellen (Umfragewert)	4,575	4,575	5	4,5
Kosten (€)	0	875	0	0

Tabelle 5-3: Werte zur Ermittlung der Zielerfüllungsfaktoren der NWA

Die Tabelle wird mit der Skala der Zielerfüllungsfaktoren kombiniert – dadurch erhält man die Zielerfüllungsfaktoren:

Alternative / Kriterium	Dojo Toolkit	Ext JS	jQuery	Prototype
Dateigröße	4	2	5	3
Eingabeelemente	3	5	0	1
Geschwindigkeit	4	4	5	1
Hilfsquellen	2	3	4	1
Kosten	5	0	5	5

Tabelle 5-4: Eingetragene Zielerfüllungsfaktoren der NWA

Diese Tabelle kombiniert man jetzt mit dem Gewichtungsfaktor:

		Dojo Toolkit		**Ext JS**		**jQuery**		**Prototype**	
	GF	ZF	NW	ZF	NW	ZF	NW	ZF	NW
Dateigröße	0,05	4	0,2	2	0,1	5	0,25	3	0,15
Eingabe-elemente	0,3	3	0,9	5	1,5	0	0	1	0,3
Geschwin-digkeit	0,3	4	1,2	4	1,2	5	1,5	1	0,3
Hilfsquel-len	0,05	2	0,1	3	0,15	4	0,20	1	0,05
Kosten	0,3	5	1,5	0	0	5	1,5	5	1,5
Nutzwertsumme:			**2,4**		**2,95**		**3,45**		**2,3**
Abkürzungen: **GF** (Gewichtungsfaktor), **ZF** (Zielerfüllungsfaktor), **NW** (Nutzwert)									

Tabelle 5-5: Nutzwerte der NWA

Die Entscheidungsalternative mit der höchsten Nutzwertsumme hat den höchsten Nutzwert. Dementsprechend ist jQuery die beste Alternative und wird für die Entwicklung eingesetzt.

6 Implementierung

6.1 Gerüst

Zunächst wurde für die HTML-Dokumente eine Dokumenttyp-Deklaration festgelegt. Eine Dokumenttyp-Deklaration legt eine Auszeichnungssprache und eine DTD (Dokumenttyp-Definition) fest:

Auszeichnungssprachen definieren die Bausteine und deren Beziehungen in einem Dokument. Als Auszeichnungssprache wird XHTML (Extensible HTML) in der Version 1.0 verwendet. XHTML verwendet als Sprachgrundlage XML (Extensible Markup Language) und hat die gleichen Elemente, Attribute und Verschachtelungsregeln wie HTML.

XHTML hat den Vorteil, dass es zu anderen XML-Standardsprachen kompatibel ist; diese Sprachen können in XHTML eingebunden werden – andersrum kann XHTML in diese Sprachen eingebunden werden. Darüber hinaus können die Dokumente mit XML-Werkzeugen betrachtet, bearbeitet und validiert werden. [11]

Eine DTD stellt Regeln auf, welche die Struktur der Dokumente vorgeben. Ein Dokument ist gültig, wenn alle Regeln eingehalten wurden. Die Dokumente sollen gültig sein, damit sie in den gängigen Browsern fehlerfrei dargestellt werden. Es gibt verschiedene Varianten für eine DTD (*strict*, *transitional*, *frameset*). Die Variante *transitional* erlaubt verschiedene Elemente und Attribute, die als *deprecated* (missbilligt) eingestuft wurden. Dazu gehören zum Beispiel Attribute wie *align* und *bgcolor*. Da solche Attribute in den bestehenden Dokumenten vorkommen, wurde die Variante *transitional* gewählt.

Die einzelnen (X)HTML-Elemente werden mit CSS (*Cascading Style Sheets*) formatiert. Mit CSS lassen sich Formateigenschaften wie Rahmen, Positionierung, Hintergrund und Schriftgröße festlegen. Die Formate werden in einer zentralen Datei definiert und können in allen Dokumenten genutzt werden. Dadurch lassen sich einheitliche

Layouts erstellen und die Formatierungen müssen nur an einer Stelle geschrieben werden.

Das Grundgerüst der Oberfläche besteht aus der Kopfleiste, dem Prozesskörper und der Fußleiste. Die Kopfleiste ist in drei nebeneinander liegende, gleichhohe Blockelemente aufgeteilt:

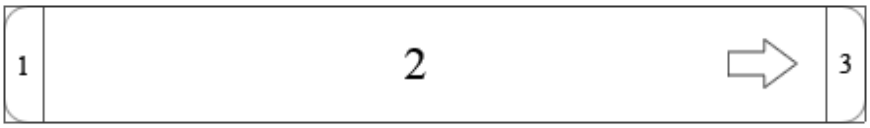


Abbildung 6-1: Aufbau der Kopfleiste

Ein Block-Element schließt in (X)HTML mehrere Elemente wie Text, Grafiken und Tabellen in einen gemeinsamen Bereich ein.
Die Block-Elemente am Rand (Abbildung 6-1 / 1 und 3) haben eine feste Breite. Das mittlere Blockelement (Abbildung 6-1 / 2) hat eine dynamische Breite – es wächst mit seinem Inhalt nach rechts. Dieses Block-Element enthält das Firmenlogo und die Prozesslinks. Damit ist sicher gestellt, dass immer alle Prozess-Links angezeigt werden.
Unter der Kopfleiste wird der Prozesskörper mit einem weiteren Block-Element eingeleitet. Die Größe des Prozesskörpers ist dynamisch und hängt vom Inhalt ab. Der Prozesskörper enthält ein Raster, an dem sich die Cards ausrichten. Dabei existiert das Raster nur theoretisch. Nur beim Ziehen und Loslassen von Cards gibt es ein sichtbares Raster – dazu mehr in Abschnitt 6.2.2. Ein Raster besteht aus einer beliebigen Anzahl von Zellen, wobei jede Zelle durch eine Reihe und Spalte definiert ist:

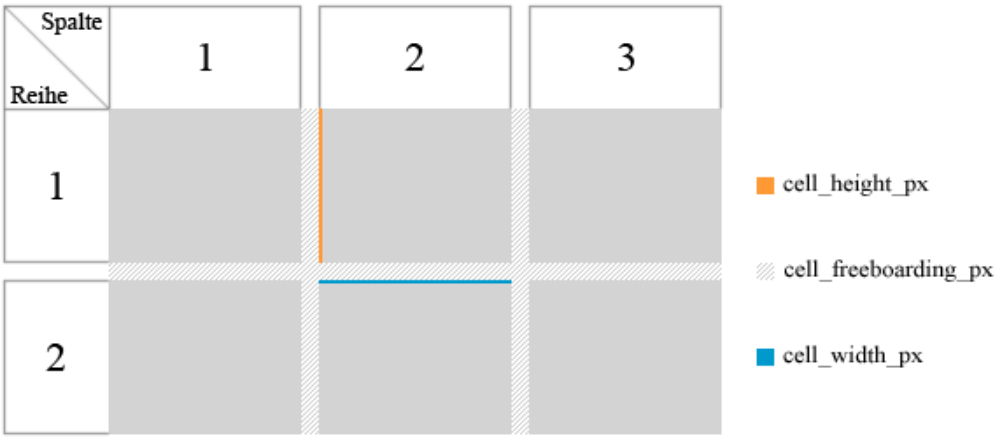


Abbildung 6-2: Aufbau eines Rasters

Die Rasterform hängt von drei Parametern ab:

- Höhe einer Zelle in Pixeln (*cell_height_px*)

- Abstand zwischen zwei Zellen in Pixeln (*cell_freeboarding_px*)
- Breite einer Zelle in Pixeln (*cell_width_px*)

Jeder dieser Werte ist allgemeingültig – jede Zelle hat die gleiche Breite, Höhe und Abstand zu einer anderen Zelle. Die Werte sind Grundlage für die Dimensionierung und Positionierung von Cards; (siehe Abschnitt 6.2.2) sie können von einem Benutzer eingestellt werden. Dadurch kann ein Benutzer alle denkbaren Größen und Anordnungen von Cards realisieren. Die Fußleiste befindet sich unter dem Prozesskörper.

Um mit dem Raster zu arbeiten, musste zuvor eine entsprechende Datenbanktabelle angelegt werden. Diese Tabelle "rasters" speichert die Raster-spezifischen Information – sie enthält eine Spalte für die Reihenanzahl (*rows*), Spaltenanzahl (*columns*), Zellenbreite (*cell_width*), Zellenhöhe (*cell_height*) und den Zellenabstand (*cell_freeboarding*). Damit einem Prozess ein Raster zugeordnet werden kann, muss eine entsprechende Relation zwischen den beiden Tabellen bestehen:

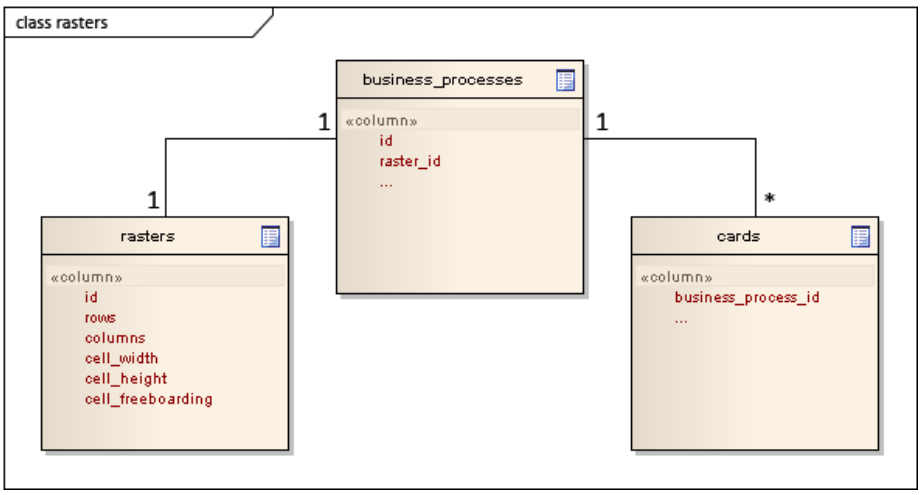


Abbildung 6-3: Datenbankrelation Raster - Cards

Zwischen der Tabelle "rasters" (Raster) und der Tabelle "business_processes" (Prozesse) wurde eine 1:1-Beziehung erzeugt. Das bedeutet jeder Prozess hat ein Raster und jedes Raster lässt sich einem Prozess zuordnen. Diese Beziehung wurde in den Rails-*models* implementiert. Dafür wurde dem *model* "BusinessProcess" folgende Zeile Code hinzugefügt:

```
has_one :raster
```

Das *model* "Raster" enthält folgende Assoziation:

```
belongs_to :business_process
```

An der 1:n-Beziehung zwischen der Tabelle “business_processes” und der Tabelle “cards” (Cards) wurde nichts geändert.

6.2 Cards

6.2.1 Dimensionierung und Positionierung

Jede Card wird mit der Funktion *draw_card()* für die Seite aufbereitet. Dabei hat jede Card eine Breite und Höhe – der Standardwert ist 1; die Breite repräsentiert die Anzahl der Spalten und die Höhe die Anzahl der Reihen auf einem Raster. Zum Beispiel hat die Card 1 in Abbildung 6-4 die Breite 2 und die Höhe 2:

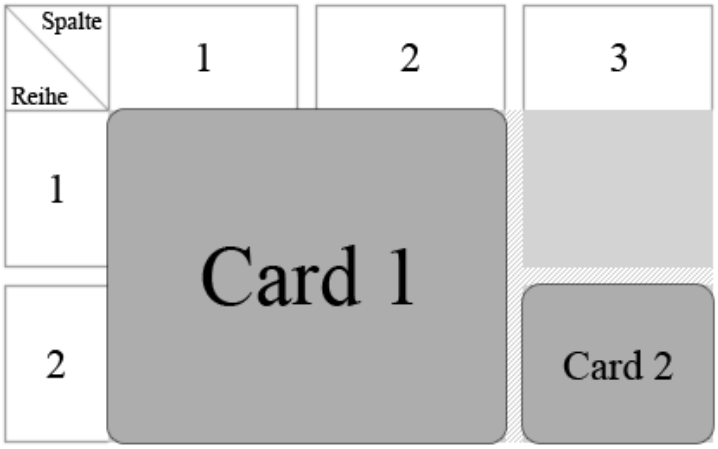


Abbildung 6-4: Cards auf einem Raster

Die Card 2 aus Abbildung 6-4 hat die Breite 1 und die Höhe 1. Wie in Abbildung 6-4 zu erkennen ist, werden die Zellenabstände mit eingeschlossen, wenn sich die Card über mehrere Reihen bzw. Spalten erstreckt. Dementsprechend berechnen sich die Anzahl der Pixel für die Breite (*card_width_px*) und Höhe (*card_height_px*) einer Card wie folgt:

card_width_px = (Card-Breite – 1) · cell_freeboarding_px + Card-Breite · cell_width_px

card_height_px = (Card-Höhe – 1) · cell_freeboarding_px + Card-Höhe · cell_height_px

Dabei ist die Höhe *card_height_px* als eine Mindesthöhe definiert: Die Card wächst nach unten, wenn die Mindesthöhe überschritten wird. Dadurch wird immer der gesamte Inhalt einer Card angezeigt – es gehen keine Informationen verloren. Da auch die

Raster-Parameter eingestellt werden können, kann jede erdenkliche Card-Größe eingestellt werden.

Die Position einer Card ist durch einen Reihen- und Spaltenwert definiert. Diese Werte beziehen sich auf eine bestimmte Zelle des Rasters. Zum Beispiel hat die Card 2 aus Abbildung 6-4 den Reihenwert 2 und den Spaltenwert 3. Eine Card breitet sich, ausgehend von dieser Zelle, nach unten bzw. nach rechts aus. In Abbildung 6-4 hat die Card 1 den Reihenwert 1, Spaltenwert 1, die Breite 2 und Höhe 2. Anhand des Tupels (Reihe, Spalte) der Zelle und den Parametern des Rasters wird eine konkrete Koordinate für eine Card errechnet. Diese Koordinate ist durch die Abstände nach oben (*card_top_px*) und links (*card_left_px*) definiert – die Abstände beziehen sich auf die linke obere Ecke des Dokuments und werden wie folgt berechnet:

card_top_px = (Card-Reihe – 1) · cell_height_px + (Card-Reihe – 1) · cell_freeboarding_px

card_left_px = (Card-Spalte –1) · cell_width_px + (Card-Spalte – 1) · cell_freeboarding_px

Eine Card wird mit diesen Abständen absolut auf der Seite positioniert. Alternativ hätte eine relative Positionierung beibehalten werden können. Die absolute Positionierung soll wegen folgender Vorteile verwendet werden: Zum einen muss nicht mehr eine Schleife (siehe Abbildung 2-3) durchlaufen werden, die für jede Zelle des Rasters prüft, ob für diese Position eine Card definiert ist. Dadurch wird Ausführungszeit eingespart. Zum anderen werden keine Block-Elemente als Platzhalter und für die Reihen benötigt. Dadurch wird das DOM schlanker, sodass die Ladezeit der Seite verkürzt wird und spätere JavaScript-Befehle die Elemente im DOM schneller finden können.

Die absolute Positionierung hat aber auch einen Nachteil: Wenn eine Card die eigene Mindesthöhe überschreitet, werden die darunterliegenden Cards nicht automatisch verschoben. Dadurch können Überlappungen wie in Abbildung 6-5 entstehen:

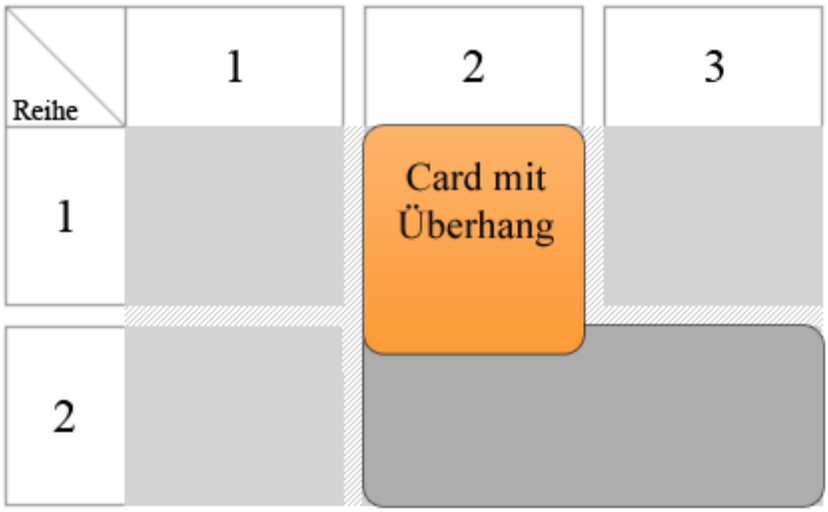


Abbildung 6-5: Card mit Überhang

Solche Überlappungen sollen nicht auftreten – trotzdem sollen die Cards weiterhin in Reihen und Spalten angeordnet werden. Dementsprechend sollen die darunter liegenden Reihen, wie in Abbildung 6-6 zu sehen, nach unten verschoben werden:

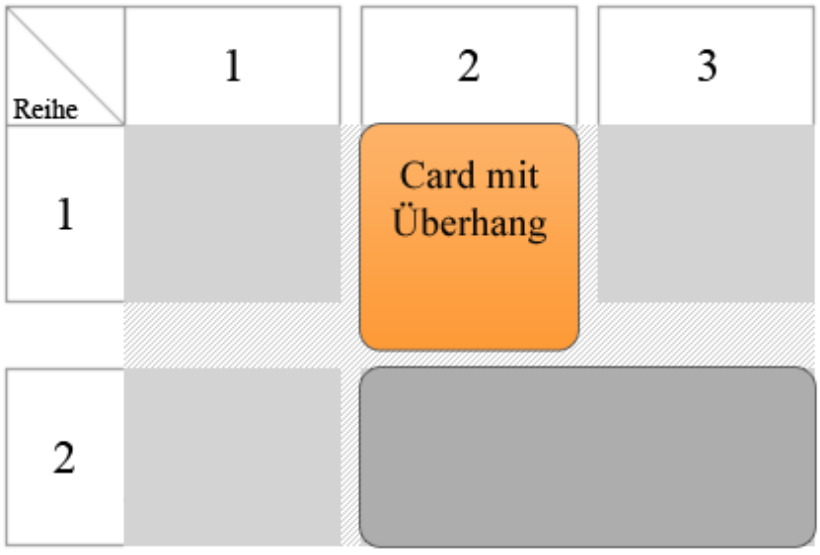


Abbildung 6-6: Neuanordnung der darunterliegenden Cards

Diese Funktionalität wurde in der *adjust_cards()*-Methode mit JavaScript und jQuery umgesetzt. Die Methode wird immer aufgerufen, nach dem sich Card-Inhalte verändert haben.

Das Prinzip der Methode ist im folgenden Aktivitätsdiagramm veranschaulicht:

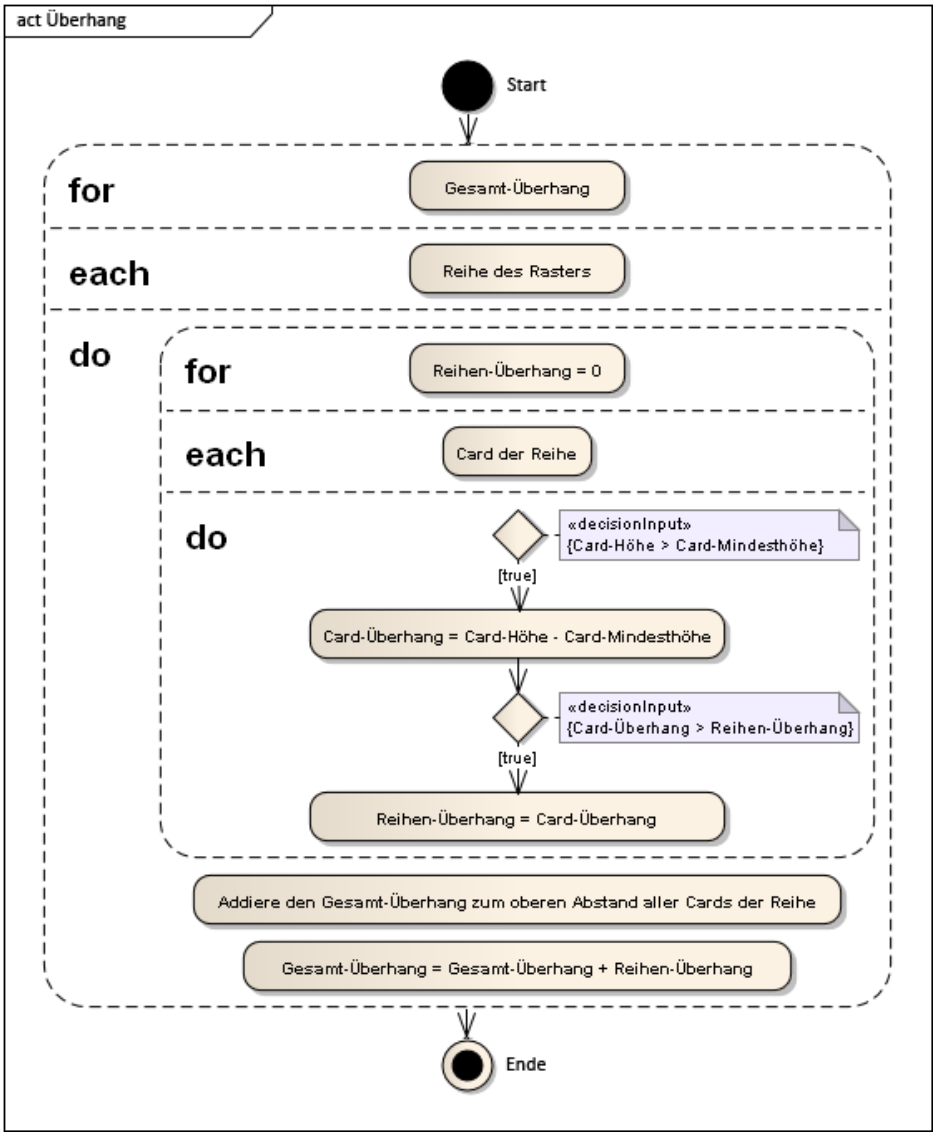
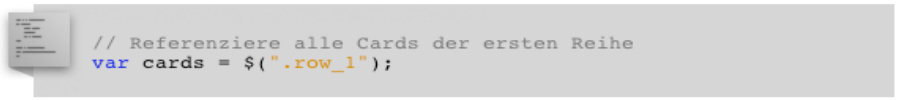


Abbildung 6-7: Aktivitätsdiagramm Überhang-Ausgleich

Die Funktion prüft für jede Reihe, ob die Cards in der Reihe einen Card-Überhang haben. Ein Card-Überhang entsteht, wenn die Card nach unten gewachsen ist, um den gesamten Inhalt darstellen zu können. Ein solcher Überhang wird als Höhe (Pixel) berechnet und repräsentiert die zusätzliche Card-Höhe. Ein Reihen-Überhang ist der höchste Card-Überhang einer Reihe – dieser wird zu dem Gesamt-Überhang addiert.
Der Gesamt-Überhang wird zu dem oberen Abstand jeder Card, der aktuellen Reihe addiert. Somit rutschen die Cards soweit nach unten, dass keine Überlappungen entstehen und trotzdem in einer Reihe bleiben.

Um diese Funktion zu implementieren, waren folgende Schritte notwendig:

(1) Jede Card muss einer Reihe zugeordnet werden können: Diese Information wird in dem *class*-Attribut des umschließenden Block-Elements einer Card hinterlegt; es hat die Syntax “row_{reihe}”, wobei {reihe} für die jeweilige Reihe ersetzt wird. Anhand dieser *classes* lassen sich mit jQuery alle Cards einer Reihe referenzieren:



```
// Referenziere alle Cards der ersten Reihe
var cards = $(".row_1");
```

(2) Der Card-Überhang muss berechnet werden können: Mit jQuery lässt sich ein Abstand nach oben bezüglich des Elternelements mit der Funktion *offsetTop()* berechnen. Ein solcher Abstand wird von einem Block-Element berechnet, welches direkt unter dem Block-Element für den Card-Inhalt liegt. Dabei haben beide Elemente dasselbe Elternelement. Dementsprechend ergibt die Berechnung immer die aktuelle Höhe des Card-Inhalts. Dieser Wert wird dann mit der Mindesthöhe des Block-Elements für den Card-Inhalt verglichen; wenn der Wert größer ist, gibt es einen Überhang. Der Überhang errechnet sich durch subtrahieren der Mindestgröße von dem Abstandswert. Darüber hinaus muss dieser Card-Überhang einer bestimmten Reihe zugeordnet werden können. Um das zu gewährleisten, enthält das Abstand-messende Block-Element eine Reiheninformation. Diese wird in dem *class*-Attribut gespeichert und hat die Form “rowmarker_{reihe}”, wobei {reihe} durch die Reihe ersetzt wird. Die entsprechende Reihe wird beim Generieren der Card wie folgt berechnet:

Reihe = Card-Reihe + Card-Höhe

Anhand dieser Information werden alle Card-Überhänge der richtigen Reihe zugeordnet: Zum Beispiel werden für eine zweite Reihe alle Card-Überhänge ermittelt, für die es ein Blockelement mit der *class* “rowmarker_2” gibt.

6.2.2 Ziehen und loslassen

Beim ziehen und loslassen können Cards mit der Maus an die verschiedenen Positionen eines Rasters verschoben werden. Das Feature wird über einen ”Prozess-bearbeiten“-Link aktiviert. Während das Feature aktiv ist, werden keine Card-Inhalte dargestellt; nur die Titelleiste der Card bleibt bestehen, damit ein Benutzer die Card zuordnen kann. Eine Card kann verschoben werden, in dem die linke Maustaste über der Card gedrückt und beim ziehen gehalten wird. Darüber hinaus wird das Raster auf der Seite dargestellt, wobei jede Rasterzelle durch ein Block-Element definiert wird. Wenn eine Card über einer Zelle losgelassen wird, wird die Card an diese Stelle verschoben. Dafür müssen genügend freie Zellen verfügbar sein oder ein Tausch der Cards möglich sein. Das Feature wurde größtenteils in JavaScript und jQuery umgesetzt und soll im Folgenden erläutert werden.

Damit eine Positionierung und Austausch von Cards möglich ist, müssen die Zellen und Cards verschiedene Selbstauskünfte enthalten. Diese Informationen sind die Grundlage für die Logik und das Speichern der Card-Positionen. Die Selbstauskünfte werden in einem *id*-Attribut jeder Zelle und Card gespeichert. Freie Raster-Zellen haben eine *id* der Form “drop_{reihe}_{spalte}”, wobei {reihe} die Reihe und {spalte} die Spalte der Zelle repräsentieren. Belegte Zellen haben eine *id* der Form “drop_{reihe}_{spalte}_{links}_{rechts}_{oben}_{unten}”; dabei werden {links}, {rechts}, {oben} und {unten} durch die Ausbreitung der Card, ausgehend von dieser Zelle, in die jeweilige Richtung ausgetauscht. In Abbildung 6-8 existiert eine Card in Reihe 1, Spalte 2 mit der Breite 2 und Höhe 2; die *id's* der Zellen würden dementsprechend so aussehen:

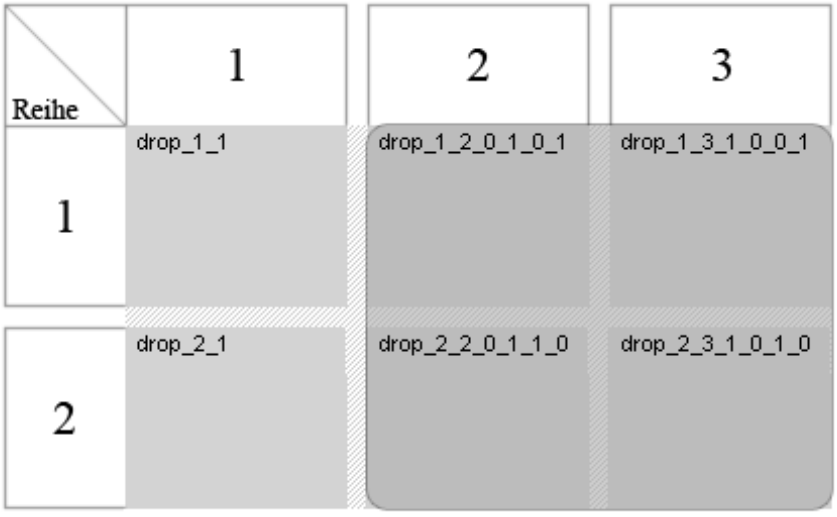


Abbildung 1-8: *id*-Attribute der Raster-Zellen

Cards besitzen eine *id* der Form “drag_{reihe}_{spalte}_{breite}_{höhe}”. Dabei repräsentiert {reihe} die Reihe, {spalte} die Spalte, {breite} die Breite und {höhe} die Höhe der Card. Also hätte die Card in Abbildung 6-8 eine *id* “drag_1_2_2_2”.
Solche *id's* werden mit jQuery und JavaScript gelesen, zerlegt, ausgewertet und wieder zusammengesetzt. Dadurch lassen sich Oberflächen-Zustände festhalten und logisch verarbeiten. Dieses Prinzip wird im folgenden Code-Block für die Ermittlung einer Zellen-Reihe und Spalte veranschaulicht:

```
// split("_") zerlegt einen String bei jedem "_" und speichert die einzelnen Werte in einem array
// - zum Beispiel generiert "x_3_2".split("_") das array {0=>"x", 1=>"3", 2=>"2"}

// zerlege die id der Zelle in ihre Einzelteile
var cell_split = $(cell).attr("id").split("_");

// parseInt() macht aus einem String eine ganze Zahl

// Reihe der Zelle
var cell_row = parseInt(cell_split[1]);
// Spalte der Zelle
var cell_column = parseInt(cell_split[2]);
```

Damit eine Card gezogen und über einer bestimmten Zelle losgelassen werden kann, müssen folgende Kriterien erfüllt sein:

(1) Eine Zelle registriert, wann eine Card über der Zelle ist und losgelassen wird. Dafür gibt es in jQuery die Funktion *droppable*, die an die Block-Elemente der Zellen gebunden wird. Die Funktion enthält eine Reihe von *callbacks*. Ein *callback* bezeichnet eine Funktion, die ausgelöst wird, wenn das Element einen bestimmten Zustand erreicht. Die *callbacks* werden benötigt, um beim Eintreten eines Zustands, einen entsprechenden Code ausführen zu können. Für die Zellen werden die *callbacks* *over* (über einer Zelle), *out* (verlassen einer Zelle) und *drop* (loslassen über einer Zelle) genutzt. Diese *callbacks* beziehen sich auf die Position des Maus-Zeigers, während eine Card gezogen wird.

(2) Die Cards müssen gezogen werden können. Dafür gibt es in jQuery die Funktion *draggable*, die an das umschließende Block-Element der Card gebunden wird. Aus der *draggable*-Funktion werden die *callbacks start* (Card gepackt) und *stop* (Card wieder losgelassen) genutzt.

Beim Auslösen des *start-callback* einer Card wird geprüft, in welchem Bereich sich der Maus-Zeiger auf der Card befindet. Dabei ergeben sich die verschiedenen Bereiche durch die darunterliegenden Zellen; die Zellenabstände werden gleichmäßig auf die Bereiche aufgeteilt. Eine solche Aufteilung ist in Abbildung 6-9 veranschaulicht:

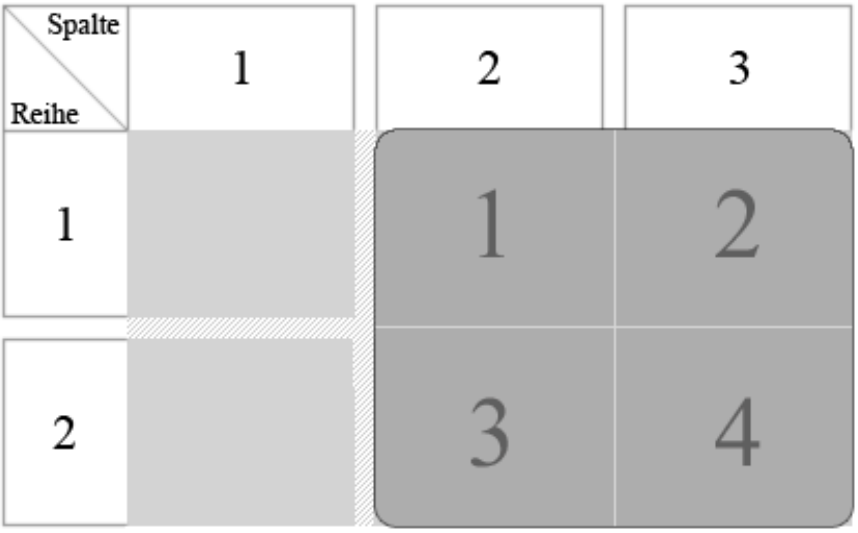


Abbildung 6-9: Bereiche einer Card

Jede Card hat also soviel Bereiche, wie sie Zellen belegt. Ein Bereich repräsentiert die Entfernung von der eigentlichen Position (Zelle) einer Card.
Die Entfernung wird durch die Anzahl von Rasterzellen nach links und oben repräsentiert. Zum Beispiel hat in Abbildung 6-9 der Bereich 1 keine Entfernung, denn er repräsentiert die Position der Card; der Bereich 2 hat die Entfernung 1 nach links; der Bereich 3 hat die Entfernung 1 nach oben; der Bereich 4 hat die Entfernung 1 nach links und 1 nach oben. Diese Entfernungen werden gebraucht, um die Card später an die richtige Position setzen zu können: Wenn der Maus-Zeiger über einer Zelle ist, kann die Card losgelassen werden; diese Zelle ist aber nicht zwangsläufig die neue Position der Card. Denn die neue Position ist auch davon abhängig, in welchem Bereich die Card mit der Maus fixiert wurde. Wird zum Beispiel in Abbildung 6-9 die Card im Bereich 4 gepackt und anschließend losgelassen, wenn der Maus-Zeiger über der Zelle in Reihe 2 Spalte 2 ist; dann soll die Card nicht in Reihe 2 und Spalte 2, sondern in Reihe 1 und Spalte 1 positioniert werden. Diese Position kann mit Hilfe der Entfernungen berechnet werden. Da die Cards nicht in Bereiche unterteilt sind, müssen die Entfernungen zunächst berechnet werden. Für die Ermittlung der Entfernung nach links werden folgende Abstände gebraucht:

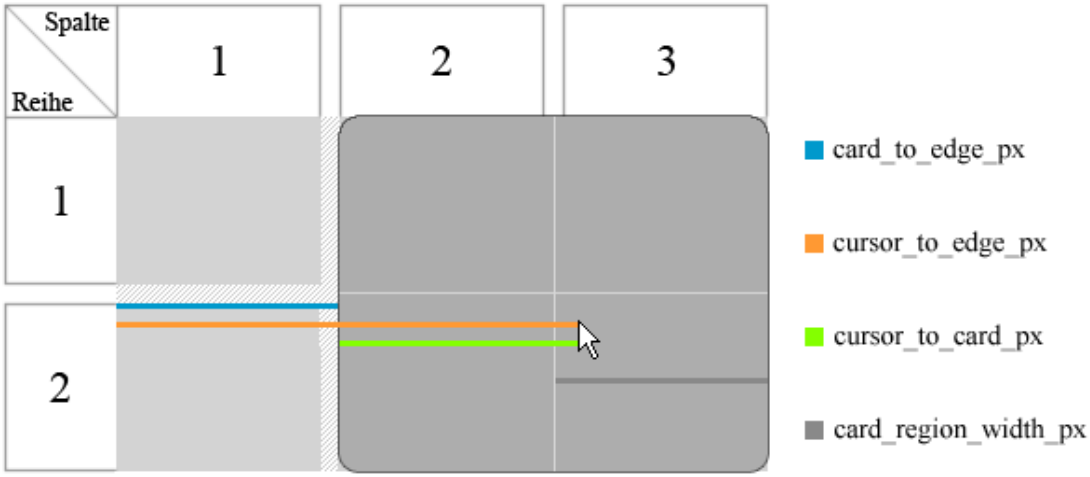


Abbildung 6-10: Abstände zum Berechnen der Card-Bereiche

Dabei bezeichnet *card_to_edge_px* den Abstand (Pixel) einer Card zum linken Bildschirmrand, *cursor_to_edge_px* den Abstand (Pixel) des Maus-Zeigers zum linken Bildschirmrand und *cursor_to_card_px* den Abstand (Pixel) des Maus-Zeigers zum linken Card-Rand. Der Abstand *card_to_edge_px* lässt sich anhand der Card-Position und der Raster-Parameter berechnen:

card_to_edge_px = (Card-Spalte – 1) · (cell_width_px + cell_freeboarding_px)

Für die Ermittlung von dem Abstand *cursor_to_edge_px* gibt es in jQuery die Funktion *page.eventX()*. Durch die Subtraktion von *cursor_to_edge_px* mit *card_to_edge_px* ergibt sich der Abstand *cursor_to_card_px*:

cursor_to_card_px = cursor_to_edge_px – card_to_edge_px

Als nächstes wird berechnet, wie viel Pixel die Card-Bereiche breit sind:

card_region_width_px = card_width_px / Card-Breite

Im Anschluss wird *cursor_to_card_px* durch *card_region_width_px* geteilt; das Ergebnis gibt die horizontale Position des Maus-Zeigers bezüglich des Card-Bereiches wieder. Zum Beispiel reicht dieser Wert in Abbildung 6-10 von 0 bis 2, weil die Card horizontal in zwei Bereiche aufgeteilt ist. Genauer gesagt geht der Wert von 0 bis 1, wenn der Maus-Zeiger in dem linken Bereich (über der Spalte 2) steht; und von 1 bis 2 wenn der Maus-Zeiger im rechten Bereich (über der Spalte 3) steht. Schließlich wird dieser Wert auf eine natürliche Zahl abgerundet und repräsentiert die Entfernung nach links (*distance_left*). Das entsprechende JavaScript dazu ist im folgenden Code-Block veranschaulicht:

```
// parseInt() überführt den Wert in eine natürliche Zahl und rundet dabei ab
distance_left = parseInt(cursor_to_card_px / card_region_width_px)
```

Die Entfernung nach oben (*distance_top*) wird nach demselben Prinzip ermittelt. In diesem Fall sind die Abstände zum oberen Bildschirmrand bzw. zum oberen Card-Rand die Grundlage der Berechnung.

Cards können nur die Position wechseln, wenn ausreichend freie Rasterzellen verfügbar sind. Darüber hinaus ist es möglich, dass eine Card die Position mit beliebig vielen anderen Cards tauscht. Dazu dürfen die Zellen, die von der Card belegt werden soll, nur abgeschlossene Cards enthalten. In Abbildung 6-11 wird das für eine Card mit der Breite 3 und Höhe 3 gezeigt:

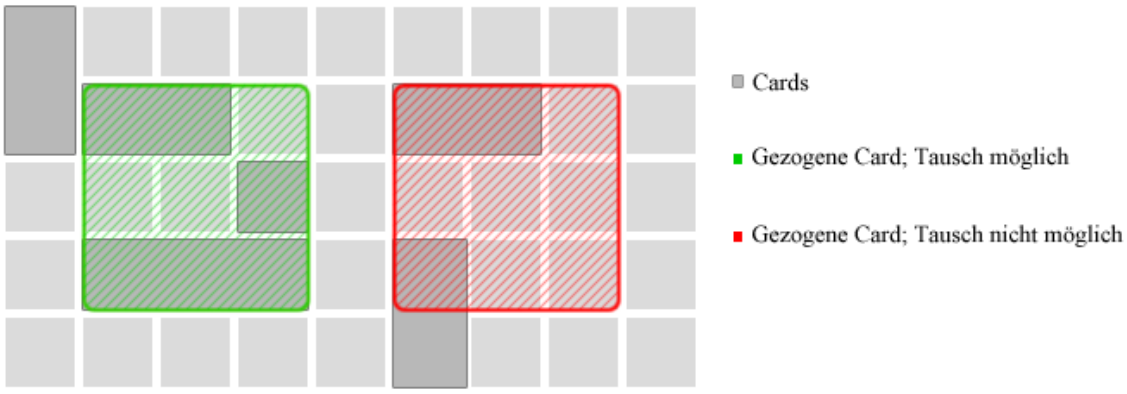


Abbildung 6-11: Tausch-Möglichkeit einer Card mit der Breite 3 und Höhe 3

Es kann also passieren, dass eine Card an eine bestimmte Position nicht verschoben werden kann. Ob die Positionierung möglich ist, wird jedes mal geprüft, wenn der *over-callback* einer Zelle ausgelöst wurde: Dann wird zunächst die Reihe (*cell_over_row*) und Spalte (*cell_over_column*) der Zelle bestimmt, über der sich der Maus-Zeiger befindet. Jetzt können die Reihe (*cell_anchor_row*) und Spalte (*cell_anchor_column*) der Zelle (cell_anchor) berechnet werden, an der sich die Card positionieren soll. Dabei werden die oben erwähnten Entfernungen genutzt:

cell_anchor_row = cell_over_row - distance_top

cell_anchor_column = cell_over_column - distance_left

Danach lässt sich die Reihe (*cell_limes_row*) und Spalte (*cell_limes_column*) der Zelle (*cell_limes*) ermitteln, welche das Ende der neuen Position von der Card repräsentiert:

cell_limes_row = cell_anchor_row + Card-Höhe

cell_limes_column = cell_anchor_column + Card-Breite

Diese Werte werden an eine Funktion *cell_checking()* übergeben, die dadurch die zu belegenden Zellen (von *cell_anchor* bis *cell_limes*) kennt. Diese Zellen werden Ziel-Zellen genannt. Die Funktion *cell_checking()* prüft rekursiv jede Zelle der Ziel-Zellen; wenn eine Zelle am Rand der Ziel-Zellen liegt, wird zunächst geprüft ob sie schon von einer Card belegt ist; falls ja wird geprüft, ob sich die Card aus den Ziel-Zellen heraus ausbreitet. Diese Prüfung ist in Abbildung 6-12 veranschaulicht:

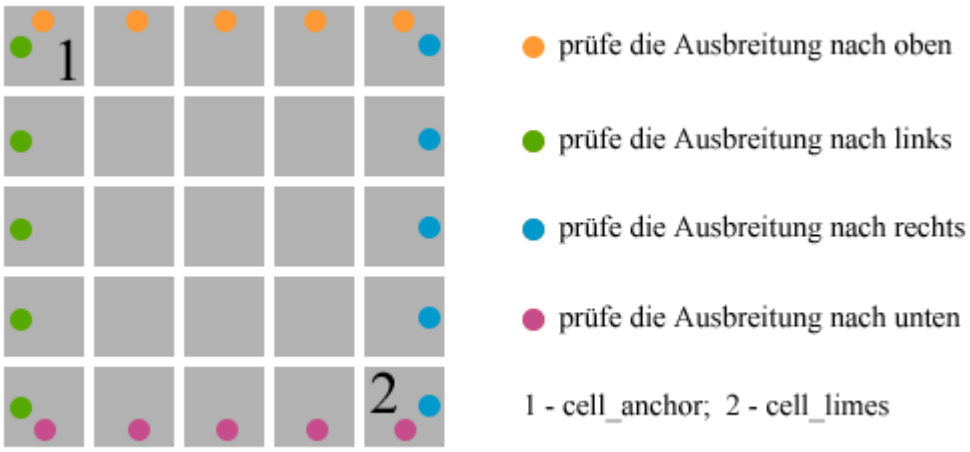


Abbildung 6-12: Überprüfung der Ziel-Zellen

Die Ausbreitung muss für jede Zelle 0 sein, sonst gibt die Funktion *false* zurück; dann kann die Card nicht auf den Ziel-Zellen positioniert werden.
Die Funktion liefert also *true*, wenn null bis beliebig viele Cards vollständig auf den Ziel-Zellen liegen; nur dann wird beim Loslassen der gezogenen Card eine entsprechende Positionierung eingeleitet.

Die Positionierung (*cards_replacing()*) folgt zunächst dem gleichen Prinzip, wie der Überprüfung der Ziel-Zellen – es wird rekursiv jede Ziel-Zelle durchlaufen.
Wenn eine Zelle belegt ist, wird die darauf liegende Card ermittelt und deren Position innerhalb der Ziel-Zellen berechnet. Anhand dieser Position wird diese Card an die entsprechende Position in den Quell-Zellen gesetzt. Die Quell-Zellen sind die Zellen, welche die gezogene Card vorher belegt hat. Sobald alle Cards aus den Ziel-Zellen in die Quell-Zellen gewandert sind, wird die gezogene Card über den Ziel-Zellen positioniert.

Ein beispielhafter Tausch von Cards wird in Abbildung 6-13 gezeigt:

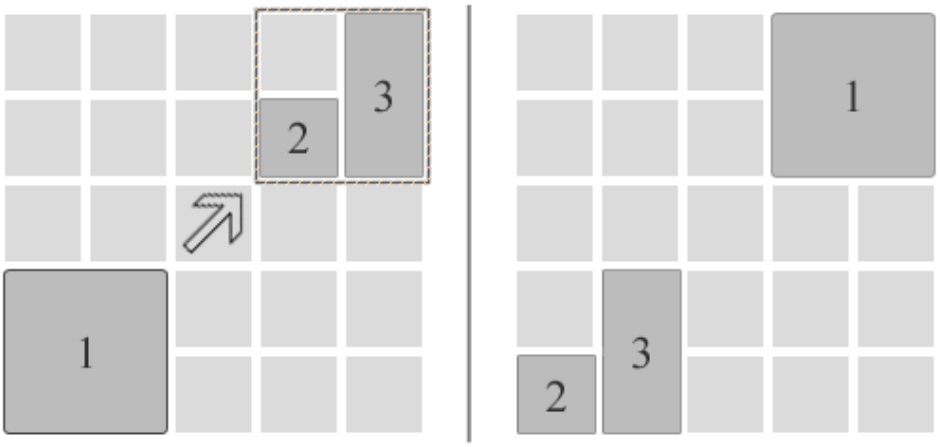


Abbildung 6-13: Tausch von Cards

Zum Schluss werden die Cards und die Zellen, entsprechend ihrer neuen Anordnung, mit neuen *id's* belegt.

6.2.3 Aufbau

Für den Aufbau einer Card werden in (X)HTML immer die gleichen Block-Elemente genutzt. Diese Block-Elemente generieren das Gerüst und Design einer Card. Für solche Block-Elemente werden die Formatierungen in einer zentralen CSS-Datei definiert. Diese Formatierungen können dann über das *class*-Attribut beliebig oft verwendet werden. Manche Block-Elemente haben aber Card-spezifische Breiten und Höhen. Diese Breiten und Höhen werden für jede Card einzeln berechnet und direkt an die Block-Elemente mit dem *style*-Attribut übergeben. Dieser Card-Aufbau wurde in einer Rails-*view* geschrieben und ist im folgenden Code-Block veranschaulicht:

```
<div class="card_border_left" style="width: <%=card_width%>px; min-height:<%=card_height%>px;">
  <div class="card_border_right" style="width: <%=card_width%>px;">
    <div class="card_border_top_left"> </div>
    <div class="card_body" style="min-height: <%=card_body_height_min_px%>px">
      <div class="space6"> </div>
      <div class="card_headbar_left"> </div>
      <div id="card_<%=card_id%>__headline" class="card_headbar" style="width: <%=card_headbar_width%>px">
        <%=card_headline%>
      </div>
      <div class="card_headbar_right"> </div>
      <div class="cleaner"> </div>
      <div id="card_<%=card_id%>__content">
        <%=card_content%>
      </div>
      <div id="card_<%=card_id%>__foot"><%=card_foot%></div>
    </div>
    <div class="card_border_top_right"> </div>
    <div class="cleaner"> </div>
    <div class="card_border_bottom_left"> </div>
    <div class="card_border_bottom" style="width:<%=card_border_bottom_width%>px"> </div>
    <div class="card_border_bottom_right"> </div>
    <div class="cleaner"> </div>
    <div class="rowmarker_<%=card_row_end%>"> </div>
  </div>
</div>
```

Wie in dem Code-Block zu erkennen ist, wird die *id* einer Card (*card_id*) in dem *id*-Attribut von verschiedenen Block-Elementen gespeichert. Anhand dieser Attribute werden die Block-Elemente später identifiziert und können einer Card zugeordnet werden. Diese Zuordnung wird gebraucht, damit gezielt Card-spezifische Inhalte mit Ajax ausgetauscht werden können. So kann der Benutzer an der Oberfläche mit den Cards arbeiten, ohne dass immer die komplette Seite neu geladen werden muss. Diese Arbeitsweise ist im folgenden Zustandsdiagramm veranschaulicht:

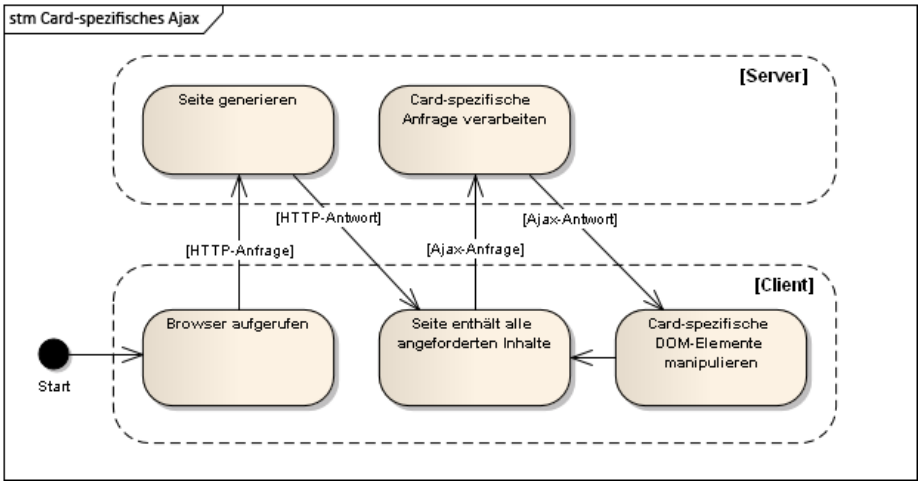


Abbildung 6-14 Zusstandsdiagramm Card-spezifisches Ajax

Jede Card hat im DOM ein eindeutiges Card-spezifisches Blockelement für das umschließende Element, den Titel (*headline*), Inhalt (*content*) und Fuß (*foot*) – das Attribut *id* hat dabei immer das gleiche Format. Solche Block-Elemente wurden möglichst tief in der (X)HTML-Verschachtelung definiert. Das soll die zu übertragene Datenmenge zwischen Server und Client so klein wie möglich halten.

6.2.4 Inkonsistente Cards

In der Applikation gibt es eine Funktion *draw_card_form()*, die alle Formulare innerhalb von Cards dynamisch generiert. Diese Funktion definiert unter anderem, wie die unterschiedlichen Formular-Felder (z.B. Textfeld, Selectbox, Checkbox) erzeugt werden. Diesen Definitionen wurde ein JavaScript-Event-Handler *onchange* hinzugefügt. Ein Event-Handler steht für ein Anwenderereignis, *onchange* für das Ereignis “bei erfolgter Änderung”. Der Event-Handler ruft die jQuery-Funktion *form_element__worked()* mit dem Parameter “Card-*id*” auf. Für ein Formular-Feld einer Card mit der *id* 1493 würde der Event-Handler entsprechend so aussehen:

```
onchange="$j(this).form_element__worked(1493)"
```

Die Funktion *form_element__worked()* fügt zunächst dem Event-Auslöser (Form-Element) eine *class* “worked” hinzu. Die *class* “worked” hat eine CSS-Formatierung und enthält einen gelben Rahmen. Durch den gelben Rahmen wird ersichtlich, welche Felder schon bearbeitet wurden. Des Weiteren wird die Funktion *card_has_unsaved_data()* mit dem Parameter “Card-*id*” aufgerufen. Diese Funktion wird auch aufgerufen, wenn ein Benutzer einen neuen Datensatz anlegt; sie ändert die Schrift-Farbe der Card-Titelleiste von weiß in gelb; diese Formatierung wird an das *style*-Attribut des entsprechenden Block-Elements übergeben; wie in Abschnitt 6.2.3 erwähnt, hat ein solches Block-Element eine Card-spezifische *id* – genauer gesagt hat es eine *id* mit dem Format “card_{Card-*id*}__headline”. Es kann also mit dem Parameter “Card-*id*” eindeutig identifiziert werden. Durch die gelbe Schrift hat ein Benutzer immer vor Augen, welche Cards ungesicherte Daten haben. Darüber hinaus wird dem umschließenden Block-Element der Card eine *class* “unsaved” gegeben; auch dieses Block-Element kann anhand seiner Card-spezifischen *id* (“card_{Card-*id*}”) eindeutig identifiziert werden. Die *class* “unsaved” wird genutzt, um zu überprüfen, ob es ungesicherte Daten auf der Oberfläche gibt.

Das soll jedes mal geprüft werden, wenn die aktuelle Seite verlassen wird; also auch wenn der Browser geschlossen wird. Für dieses Ereignis gibt es in JavaScript den Event-Handler *onbeforeunload.* Dieser Event-Handler ruft eine Funktion *closeIt()* auf. Die Funktion *closeIt()* prüft, ob die Seite ein Element mit der *class* “unsaved” enthält; falls ja wird ein Dialog “Sie haben ungesicherte Daten… Wollen Sie fortfahren?” aufgerufen. Wenn dieser bestätigt wird, wird die Aktion fortgeführt; andernfalls wird die Aktion unterbrochen und die Seite wird nicht verlassen.

Sobald ein Card-Formular gespeichert wird, wird eine Funktion *card_saved()* mit dem Parameter “Card-*id”* aufgerufen. Diese Funktion macht die Card-spezifischen Markierungen für ungesicherte Daten wieder rückgängig; es werden also die *class* “unsaved” entfernt und das *style*-Attribut der Titelleiste zurückgesetzt.

6.3 Listcards

6.3.1 Allgemeines

Die Listcard wurde auf Basis der Spezifikation und des Designs in einer Rails-*view* "list.erb.html" definiert. Diese *view* enthält die unterschiedlichen Bestandteile (Aktion-Buttons, Suchfeld, Buttons zum Seitenblättern usw.) und ruft eine Funktion *draw_card_list()* auf. Diese Funktion generiert zu einem Objekt eine entsprechende Liste; dabei wird für jeden Datensatz eine Zeile erzeugt, die aus mehreren "card_columns" besteht. Die "card_columns" repräsentieren die einzelnen Spalten einer Zeile; sie enthalten unter anderem die Daten und eine Breiten-Angabe. Um eine Basis für spätere Implementierungen zu schaffen, wurde diese Funktion leicht modifiziert:

(1) Für das Ein- und Ausblenden der Bearbeitungsebene 2 sollen Zeilen aus der Liste flüssig ein- und ausgeblendet werden können. Dafür wird eine jQuery-Funktion *animate()* genutzt. Nach einigen Tests stellte sich heraus, dass diese Funktion sich nicht auf eine Zeile (*tr*) einer HTML-Tabelle anwenden lässt; diese Tests wurden durch die jQuery-Gemeinschaft[1] bestätigt. Dementsprechend wird die Liste jetzt mit Block-Elementen erzeugt. Dafür wird für jede Zeile ein Block-Element generiert. Die Spalten werden innerhalb der Zeilen als eigenständige Block- Elemente erstellt und übernehmen die Breite der jeweiligen "card_column".

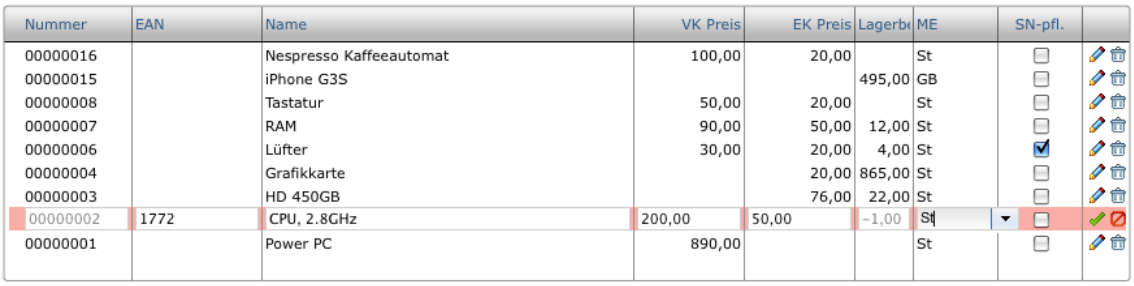
(2) Die Zeilen müssen durchnummeriert und einer Card zugeordnet werden. Dafür erhält jede Zeile eine *id* mit der Form "card_{Card-*id*}_row_{Zeilen-Nummer}"; die Zeilen-Nummer bezieht sich auf die Reihenfolge (oben nach unten) der gerade angezeigten Zeilen einer Liste. Diese Werte werden später für die Bearbeitungsebene 2, Tastaturbedienung und das fokussieren einer Zeile benötigt.

6.3.2 Bearbeitungsebene 1

Für das Öffnen einer B1 (Bearbeitungsebene 1) wird an das Ende jeder Zeile ein entsprechendes Icon gesetzt. Wenn eine B1 geöffnet wird, um einen Datensatz zu bearbeiten, wird zunächst die Funktion *load_working_planes_for_update()* mit der entsprechenden Objekt-*id* als Parameter per Ajax aufgerufen. Diese Funktion erzeugt anhand der *id* das entsprechende Objekt und gibt es an eine *view* "working_planes_update.html.erb" weiter; die Ausgabe der *view* wird mit Hilfe von jQuery in die Zeile geladen, in der die B1 geöffnet wurde. In der *view* wird, genau wie beim

[1] http://docs.jquery.com/Discussion

generieren einer Zeile, für jede “card_column” ein Block-Element mit der Breite der “card_column“ erzeugt. In dieses Block-Element wird dann das entsprechende Formular-Element geladen. Das Formular-Element wird mit der Funktion *render_card_item()* erzeugt, welche als Parameter eine “card_column” übergeben bekommt. Eine solche B1 ist in der vorletzten Zeile in Abbildung 6-15 veranschaulicht:



Nummer	EAN	Name	VK Preis	EK Preis	Lagerb	ME	SN-pfl.	
00000016		Nespresso Kaffeeautomat	100,00	20,00		St	☐	
00000015		iPhone G3S			495,00	GB	☐	
00000008		Tastatur	50,00	20,00		St	☐	
00000007		RAM	90,00	50,00	12,00	St	☐	
00000006		Lüfter	30,00	20,00	4,00	St	☑	
00000004		Grafikkarte		20,00	865,00	St	☐	
00000003		HD 450GB		76,00	22,00	St	☐	
00000002	1772	CPU, 2.8GHz	200,00	50,00	-1,00	St	☐	
00000001		Power PC	890,00			St	☐	

Abbildung 6-15: Bearbeitungsebene 1

Die Funktion *render_card_item()* gibt das Formular-Element in einer bestimmten Breite zurück. Diese Breite ist von der Breite der “card_column” und einer festgelegten Mindestbreite abhängig. Die Mindestbreiten wurden für die unterschiedlichen Formular-Element-Typen definiert; sie sollen verhindern, dass ein Formular-Feld zu schmal angezeigt wird und somit unbrauchbar wird. Dementsprechend wird in der Funktion *render_card_item()* geprüft, ob die Breite der “card_column” kleiner ist als die Mindestbreite; falls ja wird die Breite des Formular-Elements auf die Mindestbreite gesetzt, falls nicht auf die der “card_column”. Ein Formular-Element kann also breiter als seine Spalte sein. In diesem Fall wird das Element zunächst abgeschnitten; dafür sorgt die CSS-Eigenschaft *overflow:hidden* der Spalten-Elemente. Sobald ein Benutzer das Formular-Feld bearbeitet, wird aber immer das komplette Element angezeigt. Dafür wird jedem Formular-Element der B1 ein jQuery-Event *focus* zugewiesen. Dieses tritt immer auf, wenn ein Element aktiviert wird. In diesem Fall, wenn das Formular-Element angeklickt oder mit der Tastatur angesteuert wurde; dann wird dem umschließenden Block-Element des Formular-Elements eine *class* “wp1_column_relieved” zugewiesen. Für diese *class* wurde mit CSS eine absolute Positionierung definiert. Dadurch ist das Block-Element nicht mehr länger in dem Block-Element der Spalte verschachtelt und wird somit auch nicht mehr abgeschnitten. Da die *class* “wp1_column_relieved” keine Position festlegt, behält das Block-Element seine vorherige Position. Es sieht also für einen Benutzer so aus, als hätte sich nur das Formular-Element über die anderen Inhalte gelegt (zu sehen in Abbildung 6-15, Formular-Element in der Spalte “ME”). Wenn das Formular-Element wieder verlassen wird, wird es wieder in der Spalte verschachtelt und

abgeschnitten. Dafür wird jedem Formular-Element das jQuery-Event *blur* (“beim Verlassen”) zugewiesen; beim Auslösen von *blur*, wird die *class* “wp1_column_relieved” wieder entfernt.
Wenn ein Benutzer den Datensatz speichert, wird eine Funktion *update_from_wp1()* aufgerufen. Diese Funktion speichert die Daten und ersetzt die B1 wieder durch die normale Zeilendarstellung. Darüber hinaus kann der Benutzer die B1 schließen, ohne zu speichern; dafür wird der Stop-Button am Ende der Zeile geklickt, wo durch die Funktion *close_wp()* aufgerufen wird. Diese Funktion ersetzt die B1 durch die normale Zeilendarstellung.
Die B1 wird auch für das Anlegen eines neuen Datensatzes in der Listcard genutzt. Dafür wird an den Kopf der Liste eine neue Zeile gefügt. In diese Zeile wird eine B1 geladen, welche den neuen Datensatz repräsentiert. Das Laden, Speichern und Abbrechen funktioniert dabei ähnlich dem Bearbeiten eines Datensatzes. Die äquivalenten Funktionen lauten *load_wp_for_create()*, *create_from_wp()* und *wp_close_new()*; die *view* der B1 zum Anlegen lautet “working_planes_create.html.erb”.

Eine weitere Anforderung der Spezifikation ist es, eine B1 automatisch zu speichern und zu schließen, wenn eine Card-spezifische Aktion ausgeführt wird. Diese Anforderung wurde wie folgt umgesetzt:

(1) Wenn eine B1 geöffnet wurde, wird eine JavaScript-Variable *wp1_submit* angelegt. Diese Variable referenziert das HTML-Element, mit dem die B1 gespeichert und anschließend geschlossen wird; dieses Element ist das Häkchen-Icon am Ende der B1, es wird mit jQuery anhand des Bildpfades im DOM gefunden. Diese Variable wird wieder gelöscht, sobald eine B1 gespeichert bzw. geschlossen wird.
(2) Vor jeder Card-spezifischen Aktion wird geprüft, ob eine B1 offen ist; falls ja wird diese gespeichert und anschließend geschlossen. Da jede Card-spezifische Aktion mit einem Ajax-Aufruf einhergeht, wird die Überprüfung damit verknüpft. Dafür gibt es in jQuery einen *callback ajaxSend()*, der vor jedem Ajax-Aufruf ausgelöst wird. Dort wird mit der Funktion *wp_active()* geprüft, ob eine B1 geöffnet ist. Dabei prüft die Funktion, ob die Variable *wp1_submit* existiert; falls ja wird das Event *click()* der Variablen *wp1_submit* aufgerufen. Das Event *click()* simuliert ein Anklicken des Elements. Dementsprechend wird die Bearbeitungsebene gespeichert und geschlossen. Im Anschluss daran wird dann der eigentliche Ajax-Aufruf fortgesetzt.

Dieser Ablauf wird in dem folgenden Aktivitätsdiagramm gezeigt:

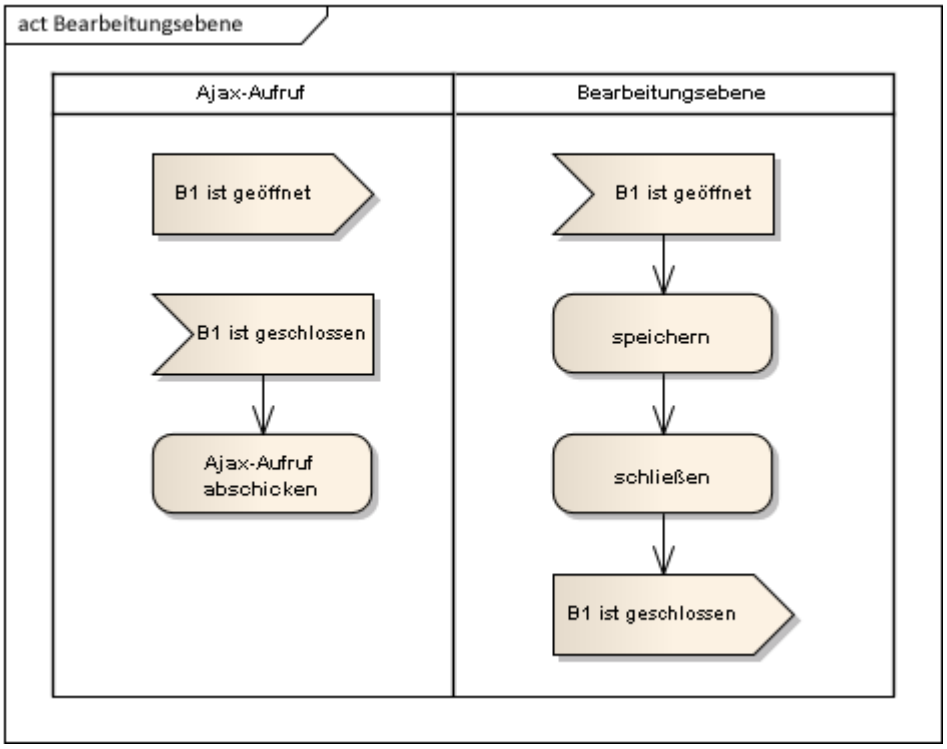


Abbildung 6-16: Aktivitätsdiagramm Bearbeitungsebene 1

Da eine B1 selbst mit einem Ajax-Aufruf geöffnet wird, können somit auch nie mehrere erste Bearbeitungsebenen gleichzeitig geöffnet sein.

6.3.3 Bearbeitungsebene 2

Die Entwicklung der B2 (Bearbeitungsebene 2) wurde in die Arbeitspakete “Konfiguration der B2” und “Arbeiten mit der B2“ aufgeteilt. Dabei war es meine Aufgabe, dass “Arbeiten mit der B2” zu implementieren. Demzufolge waren die Daten (Anzahl der Zeilen, Formular-Felder mit Position) einer B2 vorgegeben und darauf aufbauend, wurde die Funktionalität der B2 entwickelt. Eine aufgeklappte B2 ist in Abbildung 6-17 zu sehen:

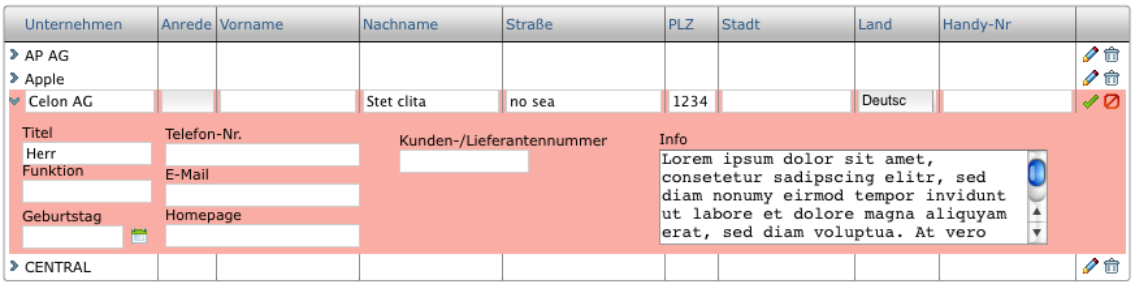


Abbildung 6-17: Bearbeitungsebene 2

Die B2 wird in der Funktion *load_working_planes_for_update()* geladen, in der auch die B1 generiert wird. Die unterschiedlichen Formular-Elemente der B2 werden in einem Block-Element mit der *id* “wp2” erzeugt. Dabei ist jedes Formular-Element in einem eigenen Block-Element verschachtelt. Diese Block-Elemente besitzen eine relative Positionierung mit den Positions-Angaben der Formular-Felder. Durch die relative Positionierung richten sich die Elemente an ihrem Mutter-Element aus. Demzufolge werden die Formular-Felder in dem Block-Element “wp2” positioniert. Das Block-Element “wp2” hat die Höhe 0 und ist somit zunächst nicht sichtbar. Die B2 ist also schon im DOM vorhanden, sobald eine B1 geladen wurde; sie kann dann ohne Verzögerung, mit dem Pfeil am Anfang der Zeile, aufgeklappt werden. Über den Pfeil kann die B2 auch direkt aufgeklappt werden – ohne vorher die B1 explizit aufzurufen. In beiden Fällen wird eine JavaScript-Funktion *open_wp2()* mit den Parametern Card-*id* und Zeilen-*id* aufgerufen. Diese Funktion markiert zunächst die Zeilen, die für die B2 ausgeblendet werden sollen; also die Zeilen, die am weitesten in der Liste von dem Datensatz entfernt sind. Diese Zeilen werden auf Basis der “auslösenden Zeile”, “Anzahl der Zeilen in der Liste” und der “Anzahl der Zeilen für die B2” ermittelt. Dabei wird die “auslösende Zeile” anhand der Zeilen-*id* bestimmt. Die “Anzahl der Zeilen in der Liste” und die “ Anzahl der Zeilen für die B2” werden über die *class* eines Block-Elements mit der *id* “card_{Card-*id*}_list_info” ermittelt. Dieses Block-Element wird beim Generieren der Liste im DOM angelegt und hat eine entsprechende *class* der Form “{Anzahl der Zeilen in der Liste }_{Höhe einer Zeile }_{ Anzahl der Zeilen für die B2}”. Die Markierung der Zeilen ist im folgenden Aktivitätsdiagramm veranschaulicht:

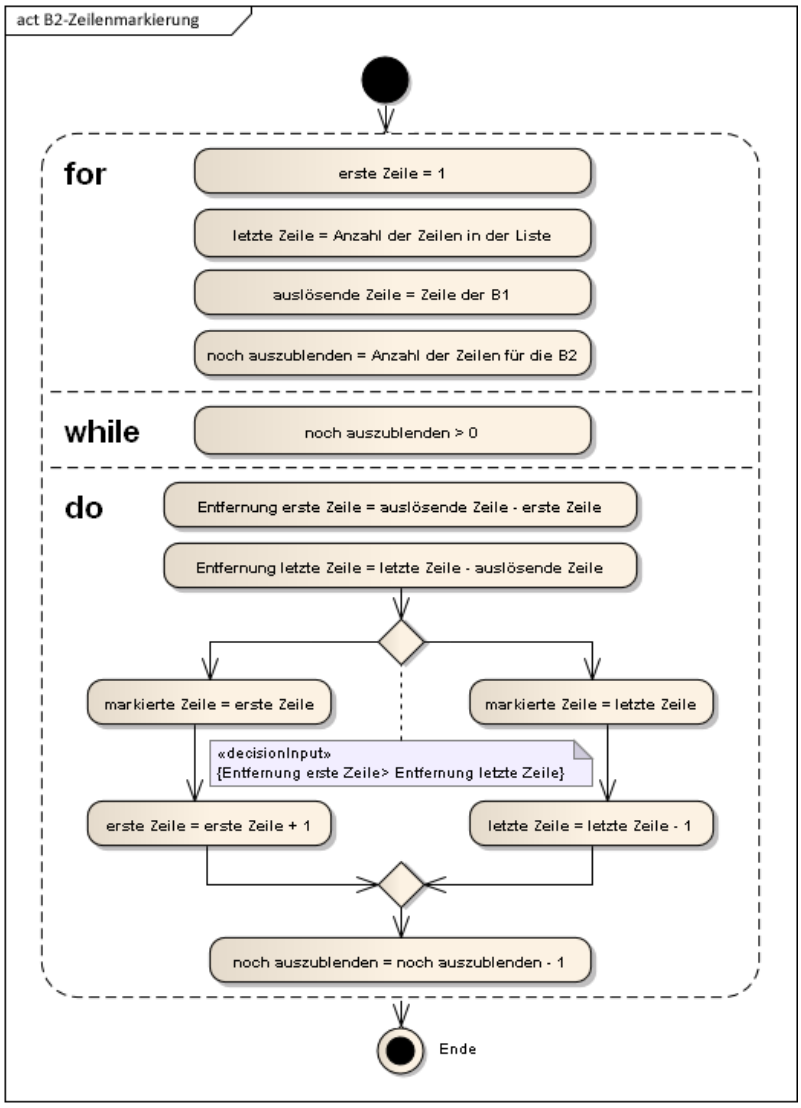


Abbildung 6-18: Aktivitätsdiagramm B2-Zeilenmarkierung

Alle markierten Zeilen besitzen eine *class* “vanish”. Diese werden jetzt mit der jQuery-Funktion *animate()* ausgeblendet; diese Funktion animiert Elemente von deren aktuellen Zustand in den angegebenen Zustand; in diesem Fall in den Zustand “Höhe 0” (unsichtbar). Des Weiteren wird die Dauer der Animation festgelegt (500 Millisekunden):

```
// Zeilen werden ausgeblendet
$j(".vanish").animate(
  { height: "0px" },
  500,
  "",
  function(){ $j(this).css("display", "none") }
);
```

Gleichzeitig wird die B2, nach dem gleichen Prinzip, mit animate() eingeblendet:

```
// Einblenden der B2
$j("#wp2").animate(
  { height: wp2_height_px + "px" },
  500
);
```

Dabei berechnet sich die Höhe (Pixel) der B2 wie folgt:

wp2_height_px = “Höhe einer Zeile” · “Anzahl der Zeilen für die B2”

Durch die Gleichzeitigkeit der Animationen, sieht es so aus, als würde die B2 die markierten Zeilen wegschieben.

Beim Schließen der B2 werden die Animationen entsprechend umgedreht: die B2 erhält wieder die Höhe 0; die markierten Zeilen bekommen wieder die Standardhöhe einer Zeile und verlieren ihre Markierung.

6.3.4 Fokussieren einer Zeile

Eine fokussierte Zeile enthält eine *class* “focus”; für diese *class* wurde in CSS eine Hintergrundfarbe definiert, sodass sich die Zeile farblich von den anderen Zeilen abhebt. Beim Generieren der Liste, wird für jede Zeile, ein *onmouseover*-Ereignis definiert. Dieses Ereignis wird immer ausgelöst, wenn der Maus-Zeiger über der Zeile ist. Dann wird eine Funktion *focus_row()* mit den Parametern Card-*id* und Zeilen-Nummer aufgerufen. Diese Funktion entfernt zunächst die *class* “focus” aus dem Dokument und fügt sie dann der Zeile hinzu.

6.3.4 Tastaturbedienung

Wie in Abschnitt 6.3.1 erwähnt, ist die Card-*id* und Zeilen-Nummer in dem *id*-Attribut der Zeile hinterlegt. Dadurch kann die vorhergehende bzw. nächste Zeile in einer Card-Liste ermittelt werden. Darüber hinaus erhalten das Suchfeld, der “Nächste-Seite”-Button und “Vorhergehende-Seite”-Button eine Card-spezifische *id* und können somit einer Card zugeordnet werden.
Um auf die Pfeiltasten-Eingaben reagieren zu können, wurde die jQuery-Funktion *keydown* genutzt. Diese Funktion wird immer aufgerufen, wenn ein Benutzer eine Taste anschlägt. Sie besitzt als Parameter das *event*-Objekt, welches das Ereignis repräsentiert; es enthält zum Beispiel Informationen darüber, von welchem Element mit welcher Taste das Ereignis aufgerufen wurde. Dadurch kann dann entsprechend im Code reagiert werden:

(1) Pfeil-Taste runter: Es wird zunächst geprüft, ob der Ereignis-Auslöser ein Suchfeld war; falls ja wird die erste Zeile der Liste fokussiert. Wenn eine Zeile den Fokus hat, wird die *id* der nächsten Zeile gebildet; falls diese Zeile existiert, bekommt sie den Fokus; falls nicht wird das *click()*-Ereignis des "Nächste Seite"-Button aufgerufen. Dadurch wird die nächste Seite geladen; im Anschluss daran erhält die erste Zeile den Fokus.

(2) Pfeil-Taste rauf: Wenn eine Zeile den Fokus hat, wird die *id* der vorhergehenden Zeile gebildet; falls diese Zeile existiert, bekommt sie den Fokus; falls nicht wird zunächst geprüft, ob ein "Vorhergehende Seite"-Button existiert; falls ja wird dessen *click()*-Ereignis ausgelöst und im Anschluss die erste Zeile markiert; falls nicht bekommt das Suchfeld den Fokus.

6.4 PRUG

Die Basis für die PRUG-Spezifikationen "Beziehungen zwischen Cards" und "Beziehungen zwischen Zeilen und Cards" sind die bereits erwähnten Card-spezifischen *id*-Attribute. Dadurch lassen sich beim Ausführen der Logik, gezielt visuelle Effekte mit jQuery auf der Oberfläche erzeugen.

Dabei wird vorrangig der jQuery-Effekt *highlight* benutzt; dieser Effekt gibt dem Block-Element zunächst eine spezifische Farbe und lässt die Farbe dann über einen bestimmten Zeitraum wieder verblassen.

Die Spezifikation "Aktiven Prozess hervorheben" wird durch eine *class* "headbar_active" realisiert. Diese *class* wird immer dem aktiven Prozess-Link zugewiesen; sie definiert eine größere Schrift und eine andere Farbe gegenüber den inaktiven Prozess-Links. Demzufolge hat ein Benutzer immer vor Augen, welcher Prozess gerade aktiv ist.

6.5 Umsetzung des Designs

Das Design der Oberfläche wird zunächst in kleinere Einzelteile (Grafiken) zerlegt. Die Einzelteile repräsentieren Block-Elemente, die so angeordnet werden, dass die Webseite wieder der Design-Vorlage entspricht.

Die Aufteilung solcher Block-Elemente ist beispielhaft für eine Card in Abbildung 6-19 dargestellt; die roten Rahmen repräsentieren jeweils ein Block-Element:

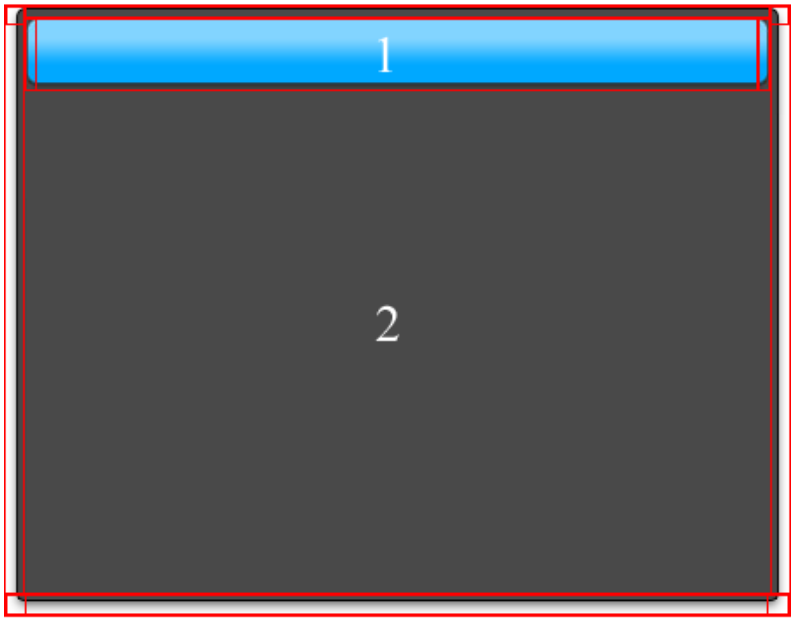


Abbildung 6-19: Aufteilung der Block-Elemente bei einer Card

Für die Block-Elemente wird eine Position, Breite, Höhe und ein Hintergrund definiert. Die Hintergründe sind die zuvor erstellten Einzelteile (Grafiken) des Designs. Das Ziel ist es, möglichst wenig und möglichst kleine Grafiken zu verwenden; dadurch werden die Ladezeiten der Webseiten verringert. Um die Grafiken möglichst klein zu halten, wurde die Möglichkeit genutzt, dass man Hintergründe in Block-Elementen wiederholen kann. Dafür gibt man in CSS ein Hintergrundbild an und in welche Richtung es sich wiederholen soll (horizontal, vertikal oder beides). Zum Beispiel hat das Block-Element 1 in Abbildung 6-19 als Hintergrundbild eine Grafik die 1 Pixel breit ist und sich horizontal wiederholt. Das entsprechende CSS ist im folgenden Code veranschaulicht:

```
div.card_headbar
{
    margin: 0px;
    padding: 0px;
    height: 42px;
    background: url(../images/bg_card_headbar.gif) top left repeat-x;
    float: left;
}
```

Eine weitere Maßnahme die Dateigrößen der Grafiken zu verringern, war die Auswahl eines geeigneten Grafikformats. In Photoshop gibt es dafür einen “Für Web und Geräte speichern”-Dialog, mit dem ein gewünschtes Grafikformat gewählt werden kann. Dabei stehen die folgenden Formate zur Auswahl:

JPEG (*Joint Photographic Experts Group*):

- Farbtiefe von 24 Bit (16,7 Millionen Farben)
- eignet sich besonders für Fotos [12]
- Dateigröße lässt sich durch Datenreduktion verringern

GIF (*Graphics Interchange Format*):

- Farbtiefe bis 8 Bit (256 Farben)
- Dateigröße lässt sich durch Farbreduzierung verringern

PNG (*Portable Network Graphics*):

- Farbtiefe bis 48 Bit
- Dateigröße lässt sich durch Farbreduzierung verringern

Darüber hinaus lässt sich, in dem "Für Web und Geräte speichern"-Dialog, für das JPEG-Format die Qualität einstellen. Dabei reduziert eine geringere Qualität die Dateigröße. Des Weiteren lässt sich für das GIF- und PNG-Format die Anzahl der Farben einstellen; umso weniger Farben eingestellt werden, desto kleiner wird die Dateigröße. Für jede Einstellung werden eine Bild-Vorschau und die Dateigröße angezeigt. Hierbei wurde eine Einstellung gewählt, mit einer möglichst kleinen Dateigröße, ohne die Qualität der Grafik wahrnehmbar nachteilig zu beeinflussen. Dabei wurde zumeist das PNG-Format mit einer Farbreduzierung gewählt, weil die meisten Hintergrundgrafiken mit wenigen Farben auskommen. Das PNG-Format hat gegenüber dem GIF-Format den Vorteil, dass es besser komprimiert und somit kleinere Dateien erzeugt. [13]

Bei einfarbigen Block-Elementen wurde gänzlich auf Grafiken verzichtet, stattdessen wurde eine Hintergrundfarbe definiert. Ein solches Block-Element ist in Abbildung 6-19 mit einer 2 markiert.

7 Evaluierung

In diesem Kapitel werden die Verfahren beschrieben, welche die Praxistauglichkeit der Oberfläche bewerten. Ein Kriterium für die Praxistauglichkeit ist die Validität der (X)HTML-Dokumente und CSS-Formatierungen. Durch die Validität werden Darstellungs- und Interpretations-Fehler der Browser vermieden. [14]
Diesbezüglich wurde untersucht, ob die Webseite nach den Regeln der Auszeichnungssprache geschrieben wurde. Wie in Abschnitt 6.1 erwähnt, wurde eine Dokumenttyp-Deklaration definiert. Das Dokument kann anhand dieser Dokumenttyp-Deklaration validiert werden. Dafür gibt es im Internet einen "Markup Validation Service"[1]. Dabei wurde die Validierung begleitend, nach dem Fertigstellen von Teilaspekten, durchgeführt. Die Validierung hat häufig kleinere Fehler aufgedeckt, die dann schnell behoben werden konnten. Des Weiteren wurde das geschriebene CSS mit dem "CSS Validation Service"[2] untersucht. Auch dabei wurden kleinere Fehler aufgedeckt, die zu Browserspezifischen Darstellungsfehlern führen können.

Eine praxisnahe, fortlaufende Evaluierung wurde durch die Mitarbeiter der onrooby GmbH betrieben: Die Geschäftsleitung benutzt die Software bereits produktiv; die Programmierer nutzen die Oberfläche, um eigene Implementierungen zu testen. Durch das ständige Arbeiten mit der Software, wird die Oberfläche dauerhaft getestet. Dabei entsteht ein Eindruck, wie sich einzelne Aspekte anwenden lassen und wo Verbesserungsbedarf besteht. Solche Eindrücke wurden in Gesprächen aufgearbeitet und daraus ggf. neue Aufgaben generiert. Hierbei ist die Grundlage der fortlaufenden Evaluierung eine Versionsverwaltung. Eine Versionsverwaltung ist ein System, mit dem Änderungen an Dokumenten und Dateien verwaltet werden. Der Entwicklungsprozess der Software wurde versioniert, sodass alle Mitarbeiter einen aktuellen Stand der Software beziehen können.

Darüber hinaus wurde ein Fragebogen (siehe Anhang H) für die Käufer der Software erstellt. In diesem Fragebogen werden der Nutzen und die Umsetzung der einzelnen Aspekte der Oberfläche bewertet. Durch die Auswertung solcher Fragebögen, soll überprüft werden, wie tauglich und zufriedenstellend die Oberfläche für den Endkunden ist. Daraus können ggf. neue Aspekte und Verbesserungen abgeleitet werden.

Der Fragebogen wurde in Einleitung, Hauptteil und Endteil gegliedert. In dem Einleitungsteil werden allgemeine Hinweise gegeben, welche eventuelle Fragen des Befragten beantworten. Der Hauptteil enthält 17 Fragen, 3 Felder für Teilnehmer-bezogene Daten und ein Kommentarfeld. Dabei wurden die Fragen in zusammenhängende Themenbereiche gegliedert. [15]

Die Teilnehmer-bezogenen Fragen wurden an das Ende des Hauptteils gesetzt, da solche Angaben den Teilnehmer langweilen könnten: Der Befragte könnte schon am Anfang das Interesse verlieren und das Ausfüllen abbrechen. Am Ende des Fragebogens werden solche Angaben eher gemacht, damit das Ausfüllen nicht umsonst war. Der Endteil enthält einen Dank für das Ausfüllen des Fragebogens. Ein solcher Dank kann die Motivation zur schnellstmöglichen Rücksendung steigern. [16]

Der Fragebogen wird Anfang 2010 verteilt, nach dem die Software fertiggestellt und veröffentlicht wurde. Im Anschluss daran, werden die Bögen dann ausgewertet.

8 Zusammenfassung

8.1 Ergebnisse

Der theoretische Teil beschäftigt sich zunächst mit dem Programm onrooby. Dazu gehören eine Einführung in WWS und die Vorstellung des Programms. Des Weiteren wird das bestehende Konzept für die Benutzeroberfläche von onrooby erläutert.
Im nächsten Teil werden Grundlagen zum Verständnis der Arbeit vorgestellt. Hierzu gehören eine Einführung in das Framework Rails, eine Einführung in JavaScript und das DOM, eine Einführung in Ajax und die Erläuterung der Nutzwertanalyse.
Im folgenden Teil wurde, auf Grundlage des Konzepts, die Benutzeroberfläche spezifiziert. Zu diesem Punkt gehören: Aufbau und Design der Oberfläche; Aufbau und Funktionen von Cards, Listcards und Detailcards, sowie die einzelnen Bestandteile von PRUG. Nachfolgend wurde, auf Basis einer Nutzwertanalyse, eine geeignete JavaScript-Bibliothek ausgewählt. Dabei stellte sich heraus, dass jQuery am geeignetsten ist.
Als nächstes wurde, auf Grundlage der Spezifikation, die Oberfläche implementiert. Dabei wurden folgende Teilaspekte umgesetzt:

- Installation von jQuery
- Umsetzung des Designs
- Neustrukturierung und Überarbeitung der (X)HTML-Dokumente für das Oberflächen-Gerüst, Card-Gerüst, List- und Detailcards
- Formatieren der (X)HTML-Elemente in einer zentralen CSS-Datei
- Entwicklung einer Grundlage für das Positionieren, Dimensionieren, Ziehen und Loslassen, sowie die Größenänderung mit der Maus einer Card; hiermit ist das in Abschnitt 6.1 vorgestellte Raster gemeint
- Erneuerung des Positionieren und Dimensionieren von Cards, auf Grundlage des Rasters
- der sehr umfangreiche Teil meiner Arbeit: Die Umsetzung des Ziehen und Loslassen von Cards
- Entwicklung einer Lösung, den Benutzer vor dem Verlust nicht gespeicherter Daten zu schützen
- Überarbeitung des Generierens einer Liste
- Implementierung der beiden unterschiedlichen Bearbeitungsebenen

- Umsetzen weiterer Funktionen einer Listcard; dazu gehören das Fokussieren einer Zeile und die Tastaturbedienung
- Card-übergreifende Beziehungen wurden visualisiert

In einem weiteren Kapitel wird die Evaluierung der Benutzeroberfläche erläutert. Dabei wurde unter anderem ein Fragebogen für den Endkunden angefertigt.

8.2 Ausblick

Die Benutzeroberfläche soll vervollständigt und erweitert werden. In naher Zukunft werden zunächst die offenen Punkte der Spezifikation umgesetzt:

- Größenänderung mit der Maus (siehe Abschnitt 4.3.3)
- Workflow-Recorder (siehe Abschnitt 4.6.3)

Darüber hinaus werden die Fragebögen ausgewertet, sobald diese ausgefüllt vorliegen. Auf Basis der Auswertung werden dann ggf. Anpassungen und Erweiterungen umgesetzt.

Ferner soll die Benutzeroberfläche noch erweitert werden. Dazu sind im Laufe der Arbeit schon Ideen entstanden:

- Ein Benutzer soll zwischen Oberflächen-Designs wählen können. Dabei unterscheiden sich die Designs vor allem in den Farben.
- Card-spezifische Hintergründe: Für die verschiedenen Cards soll es bestimmte Hintergründe geben. Zum Beispiel hat eine Personen-Listcard im Hintergrund Menschen abgebildet.
- Die Formular-Elemente sollen mit der Maus angeordnet werden können. Dieses Feature funktioniert ähnlich, wie das Ziehen und loslassen von Cards. Dadurch lassen sich Detailcards und die zweite Bearbeitungsebene komfortabler konfigurieren.
- Die Listen sollen einfacher zu konfigurieren sein: Das Einstellen der Spaltenbreiten soll mit einem Ziehen und Loslassen der Maus zu bewältigen sein.
- Es soll eine dritte Bearbeitungsebene geben; diese Ebene wird mit einem “mehr”-Link geöffnet, welcher sich am Ende der zweiten Bearbeitungsebene befindet. In dieser Ebene können weitere Formular-Elemente definiert werden, für die in der zweiten Ebene kein Platz mehr ist. Die Ebene erscheint unter der zweiten Ebene und

legt sich über die anderen Inhalte. Sie wird automatisch geschlossen, wenn die B2 geschlossen wurde. Darüber hinaus kann sie mit einem “weniger”-Link geschlossen werden.

- Die Tastaturbedienung soll erweitert werden; es sollen folgende Aktionen mit der Tastatur ausgeführt werden können: Neuen Datensatz anlegen; Datensatz speichern; Datensatz löschen; Drucken; zwischen der Kopfleiste und dem Prozesskörper wechseln; einen bestimmten Prozess auswählen; zwischen den Elementen innerhalb einer Card wechseln.

Anhang A - Nutzwertanalyse bezüglich der Eingabeelemente

Entscheidungsalternativen

- Dojo Toolkit
- Ext JS
- jQuery
- Prototype

Bewertungskriterien

Combobox

Mindestanforderungen:

- *Auswählen eines Wertes aus einer Liste*
- *Eingabe im Textfeld (Filterung der Liste)*

Weitere Anforderungen:

- *Autovervollständigung*
- *JS-Validierung (wenn kein gültiger Wert eingetragen ist)*
- *Optimale Platzausnutzung der Liste*
- *Tastaturbedienung*
- *Benutzungsfreundlichkeit*
- *schöne Optik*

Datepicker

Mindestanforderung:

- *Auswählen eines Datums aus einem Kalender*
- *Kalender zeigt Kalenderwoche an*

Weitere Anforderungen:

- *Internationalisierung*
- *Benutzungsfreundlichkeit*
- *schöne Optik*

Timepicker

Mindestanforderung:

- *Auswählen einer Zeit*

Weitere Anforderungen:

- *Internationalisierung*
- *einfache Bedienung*
- *schöne Optik*

Gewichtungsfaktoren

Kriterium	Combobox	Datepicker	Timepicker	Gewicht	Gewichtsfaktor
Combobox		1	1	2	0,25
Datepicker	1		1	2	0,25
Timepicker	1	1		2	0,25
			Gewichtssumme:	8	

Skala der Zielerfüllungsfaktoren

Skala / **Kriterium**	0	1	2	3	4
	ungenügend	mangelhaft	ausreichend	befriedigend	gut
Combobox	Mindestanforderungen nicht erfüllt	0 erweiterte Anforderungen erfüllt	1 bis 2 erweiterte Anforderungen erfüllt	3 bis 4 erweiterte Anforderungen erfüllt	5 bis 6 erweiterte Anforderungen erfüllt
Datepicker	Mindestanforderungen nicht erfüllt	0 erweiterte Anforderungen erfüllt	1 erweiterte Anforderungen erfüllt	2 erweiterte Anforderungen erfüllt	3 erweiterte Anforderungen erfüllt
Timepicker	Mindestanforderung nicht erfüllt	0 erweiterte Anforderungen erfüllt	1 erweiterte Anforderungen erfüllt	2 erweiterte Anforderungen erfüllt	3 erweiterte Anforderungen erfüllt

Nutzwerte und Ergebnis

Alternative / Kriterium	Dojo Toolkit	Ext JS	jQuery	Prototype
Combobox	4	4	0	0
Datepicker	0	4	0	0
Timepicker	4	4	1	3

		Dojo Toolkit		**Ext JS**		**jQuery**		**Prototype**	
	Gewichtungsf.	Zielerfüllungsf.	Nutzwert	Zielerfüllungsf.	Nutzwert	Zielerfüllungsf.	Nutzwert	Zielerfüllungsf.	Nutzwert
Combobox	0,25	4	1	4	1	0	0	0	0
Datepicker	0,25	0	0	4	1	0	0	0	0
Timepicker	0,25	4	1	4	1	1	0,25	3	0,75
		Summe:	**2**		**3**		**0,25**		**0,75**

Anhang B - Messdaten Slickspeed-Geschwindigkeitstest

	Internet Explorer 8		Firefox 3.5.4		Safari 4.0.3		**Ø**
Dojo Toolkit	60 ms	66 ms	94 ms	82 ms	47 ms	43 ms	**65,33** ms
Ext Js	52 ms	55 ms	104 ms	90 ms	49 ms	39 ms	**64,83** ms
jQuery	50 ms	49 ms	74 ms	71 ms	33 ms	29 ms	**51** ms
Prototype	66 ms	77 ms	134 ms	122 ms	74 ms	66 ms	**89,83** ms

Anhang C - Quelltexte (XHTML/ERB)

Card-view

```
<div class="card_border_left" style="width: <%=card_width%>px; min-height:<%=card_height%>px; height:auto !important; height:<%=card_height%>px;">
  <div class="card_border_right" style="width: <%=card_width%>px;">
    <div class="card_border_top_left"> </div>
    <div class="card_body" style="min-height: <%=card_body_height_min_px%>px">
      <div class="space6"> </div>
      <div class="card_headbar_left"> </div>
      <div id="card_<%=card_id%>__headline" class="card_headbar" style="width: <%=card_headbar_width%>px">
        <%=card_headline%>
      </div>
      <div class="card_headbar_right"> </div>
      <div class="cleaner"> </div>
      <div id="card_<%=card_id%>__content">
        <%=card_content%>
      </div>
      <%=card_attachments__upload%>
      <ul id="attachments"><%=card_attachments%></ul>
      <div id="card_<%=card_id%>__foot"><%=card_foot%></div>
    </div>
    <div class="card_border_top_right"> </div>
    <div class="cleaner"> </div>
    <div class="card_border_bottom_left"> </div>
    <div class="card_border_bottom" style="width:<%=card_border_bottom_width%>px"> </div>
    <div class="card_border_bottom_right"> </div>
    <div class="cleaner"> </div>
    <div class="rowmarker_<%=card_row_end%>" style="height: 1px; line-height: 1px; font-size: 1px; background: #ffffff;"> </div>
  </div>
</div>
```

Listcard-view

```
  <div class="card_neck" style="width:<%=card_table__width%>px">
    <div class="card_neck__left" id="card_<%=card.id%>__neck_left">
      <%=link_to_remote image_tag("plus.gif"), :url => {:controller => card.app_model_name.tableize, :action => "load_working_planes_for_create", :id => card.id}%>

      <%
      controller_class = "#{card.app_model_name.pluralize}Controller".constantize
      if controller_class.instance_methods.include? 'list_report'
      %>
        <%= link_to image_tag("print.gif", :class => "card_icon"), {:controller => card.app_model_name.tableize, :action => 'list_report', :card_id => card.id, :bpsession => @bpsession_id} -%>
      <%
      end
      %>
      <!-- Rausnehmen? Lars nochmal fragen
      <%#= link_to_remote "[Zuletzt geändert]", :url => { :controller => card.app_model_name.downcase.pluralize, :action => 'card_list_presets', :id => card.id, :what => '[last10]'}, :method => :get %>
      <%#= link_to_remote "[Alle mit »p«]", :url => { :controller => 'cards', :action => 'suchfeld_liste', :id => card.id, :search => 'p'}, :method => :get %>
      -->
    </div>
    <div class="card_neck__right" id="card_<%=card.id%>__neck_right">
      <% if @bpi.collection[card.id]
        textfeldvalue = @bpi.collection[card.id].strip
      else
        textfeldvalue = ""
      end %>
      <%= submit_tag " ", :class => "search_button" %>
```

```
25        <%= text_field_tag "suchfeldliste_#{card.id}", "", {:class => "search_text",
:value => textfeldvalue} -%>
26        <%= observe_field "suchfeldliste_#{card.id}", :frequency => 0.5, :url =>
{:controller => 'cards', :action => 'suchfeld_liste', :id => card.id, :bpsession =>
@bpsession_id, :only_path => false}, :with => "'search=' + encodeURIComponent(value)"
%>
27      </div>
28      <div class="cleaner"> </div>
29    </div>
30
31 <ul id="card_<%=card.id%>__validation_errors" class="validation_errors"> </ul>
32 <div class="data" style="width:<%=card_table__width%>px;"
id="card_<%=card.id%>__list">
33      <%= draw_card_list card %>
34 </div>
35 <div id="card_<%=card.id%>__pagination" class="card_paginator"
style="width:<%=card_table__width%>px;">
36    <%= will_paginate_ajax card if @bpi.card[card.id].respond_to?('total_pages') &&
@bpi.card[card.id].total_pages > 1%>
37 </div>
```

Anhang D - Quelltexte (Ruby)

```ruby
def draw_card card
  content_for :head do
    javascript_tag("recentlyRenderedCards.push(#{card.id});")
  end
  # Card-Abmessungen berechnen
  card_width_px = get_card_width_px card.width unless card.width.blank?
  card_height_px = get_card_height_px card.height
  card_body_height_min_px = card_height_px - CARD_BORDER_BOTTOM_HEIGHT +
(card.height - 1) * CARD_FREEBOARD
  card_headbar_width_px = get_card_headbar_width_px card_width_px   unless
card.width.blank?
  card_border_bottom_width_px = get_card_border_bottom_width_px card_width_px  un-
less card.width.blank?
  card_row_end = card.pos_row + card.height - 1

  card_headline = ""
  card_content = ""
  card_attachments__upload = ""
  card_attachments = ""
  card_foot = ""

  unless card.cardpartial.blank?
    # Card wird aus einem fertigen Partial geladen
    card_list_width_px = get_list_should_width_px card_width_px
    card_list_height_px = get_list_height_px card_height_px
    render :partial => "cardviews/#{card.cardpartial}", :locals => {:card => card,
:card_width => card_width_px, :card_height => card_height_px, :card_headbar_width =>
card_headbar_width_px, :card_border_bottom_width => card_border_bottom_width_px,
:card_headline => card_headline, :card_content => card_content, :card_id => card.id,
:card_attachments__upload => card_attachments__upload, :card_attachments =>
card_attachments, :card_foot => card_foot, :card_row_end => card_row_end,
:card_body_height_min_px => card_body_height_min_px, :card_table__width =>
card_list_width_px, :card_table__height => card_list_height_px}
  else
    # Card-Attachments erfassen
    if @bpi.card[card.id].allow_attachments
      card_attachments__upload += render :partial => 'cards/card_upload', :locals =>
{:card => card} if @bpi.card[card.id].id
      @bpi.card[card.id].attachments.each do |attachment|
        card_attachments +=render(:partial => 'cards/attachment', :object => attachment)
      end
    end
    # Listencards
    if card.cardtype == Card::CARDTYPE_LISTE
      card_list_width_px = get_list_should_width_px card_width_px
      card_list_height_px = get_list_height_px card_height_px

      # Ist die Card eine Subcard und hat keine Eltern-Card?
      if card.subcard
        unless @bpi.card[card.parent_id]
          card_headline = card.app_model.human_name
          render :partial => "cards/card", :locals => {:card_width => card_width_px,
:card_headbar_width => card_headbar_width_px, :card_border_bottom_width =>
card_border_bottom_width_px, :card_headline => card_headline, :card_content =>
card_content, :card_id => card.id, :card_attachments__upload =>
card_attachments__upload, :card_attachments => card_attachments, :card_foot =>
card_foot, :card_row_end => card_row_end, :card_body_height_min_px =>
card_body_height_min_px}
        else
          if !card.association_name.blank? && !card.parent.blank?
            list_size = eval('@bpi.card[card.parent.id].' + card.association_name + '.size')
            card_headline = "#{@bpi.card[card.parent.id].label}
#{card.parent.app_model.human_attribute_name card.association_name} #{'(' + list_size.to_s +
')' if list_size > 0}"
          else
            card_headline = card.app_model.human_name :count => 2
          end
        end
      else
        card_headline = card.app_model.human_name :count => 2
      end
```

```
 53       card_content += render :partial => "cards/list", :locals => {:card => card,
:card_table__width => card_list_width_px, :card_table__height => card_list_height_px}
 54     elsif card.cardtype == Card::CARDTYPE_DETAIL
 55       # Einzel-Cards
 56       if @bpi.card[card.id].id
 57         card_headline = "#{card.app_model.human_name} #{@bpi.card[card.id].label}"
 58         card_content = render :partial => "card_forms/detail_sub", :locals => {:card =>
card}
 59         card_foot = "card_id: #{card.id}"
 60         #render :partial => "cards/card", :locals => {:card_width => card_width,
:card_headbar_width => card_headbar_width, :card_bottomslice_width =>
card_bottomslice_width, :card_headline => card_headline, :card_content => {:partial =>
"cards/detail_sub"}, :card_id => card.id, :card_attachments__upload =>
card_attachments__upload, :card_attachments => card_attachments, :card_foot =>
card_foot}
 61       else
 62         # Card zeigt nur eine Suchmaske fuer sein app_model
 63         card_headline = card.app_model.human_name
 64         if card.subcard
 65           card_content = link_to_remote image_tag("plus.gif"), :url => {:controller =>
card.app_model_name.tableize, :action => 'new_card', :id => card.id}
 66         else
 67           card_content = suchfeld_objekte card.app_model,
card.app_model_name.humanize, 'processes', "/#{card.app_model_name.tableize}/",
"_card_#{card.id}"
 68         end
 69       end
 70     end
 71     render :partial => "cards/card", :locals => {:card_width => card_width_px,
:card_height => card_height_px, :card_headbar_width => card_headbar_width_px,
:card_border_bottom_width => card_border_bottom_width_px, :card_headline =>
card_headline, :card_content => card_content, :card_id => card.id,
:card_attachments__upload => card_attachments__upload, :card_attachments =>
card_attachments, :card_foot => card_foot, :card_row_end => card_row_end,
:card_body_height_min_px => card_body_height_min_px}
 72   end
 73 end
 74
 75 # Helper zur Ausgabe einer Form einer Card
 76 def draw_card_form(card,f)
 77   ret = "<ul id=\"card_#{card.id}__validation_errors\" class=\"validation_errors\"> </ul>"
 78   card.card_fields.each do |field|
 79     label = "<label>#{card.app_model.human_attribute_name field.app_field_name}</label>"
 80     info = ""
 81     unless field.info.blank?
 82       info = image_tag("info.gif", :alt => field.info, :title => field.info)
 83     end
 84
 85     case field.app_field_type(@bpi.card[card.id])
 86       when :string:
 87         href = ""
 88         field_value = eval("@bpi.card[card.id].#{field.app_field_name}")
 89         #width = '229px'
 90         if field.app_field_name.include?('email') && field_value =~ /[0-9a-zA-Z.-]+@[0-
9a-zA-Z.-]+\.[a-zA-Z]/
 91           href = mail_to field_value, image_tag('12-mail.gif')
 92           #width = '211px'
 93         elsif field.app_field_name.include?('url') && field_value =~ /https?:\/\/[0-
9a-zA-Z.-]+\.[0-9a-zA-Z]/
 94           href = link_to image_tag('12-arrow-right.gif'), field_value, {:target => 'blank'}
 95           #width = '211px'
 96         elsif field.app_field_name.include?('company_name') && !field_value.blank?
 97           href = link_to image_tag('google.gif'),
"http://www.google.de/search?q=#{field_value.gsub(/ /, '+')}", {:target => 'blank'}
 98           #width = '211px'
 99         end
100
101         ret += "<p>#{label}#{f.text_field field.app_field_name, :onchange =>
"$j(this).form_element__worked(#{card.id})", :disabled => ('disabled' if field.visible != 2)}\n#{info}"
102         ret += href
103         ret += "</p>"
104
105       when :integer:
```

```
106           ret += "<p>#{label}#{f.text_field field.app_field_name, :onchange =>
"$j(this).form_element__worked(#{card.id})", :disabled => ('disabled' if field.visible !=
2)}\n#{info}</p>"
107
108         when :decimal:
109           ret += "<p>#{label}#{f.amount_field field.app_field_name, :onchange =>
"$j(this).form_element__worked(#{card.id})", :disabled => ('disabled' if field.visible !=
2)}\n#{info}</p>"
110
111         when :float:
112           ret += "<p>#{label}#{f.amount_field field.app_field_name, :onchange =>
"$j(this).form_element__worked(#{card.id})", :disabled => ('disabled' if field.visible !=
2)}\n#{info}</p>"
113
114         when :boolean:
115           ret += "<p>#{label}#{f.check_box field.app_field_name, :onchange =>
"$j(this).form_element__worked(#{card.id})", :disabled => ('disabled' if field.visible !=
2)}\n#{info}</p>"
116
117         when :date:
118           ret += "<p>#{label}"
119           if field.visible != 2
120             # TODO: Date korrekt auch in Feldern anzeigen
121             #ret += "#{f.text_field field.app_field_name, :onchange =>
"$j(this).form_element__worked(#{card.id})", :disabled => 'disabled'}"
122           else
123            token = rand(36**8).to_s(36)
124            datepicker_id = "#{card.id}_datepicker_#{field.app_field_name}_#{token}"
125            value = eval("f.object.#{field.app_field_name}")
126            if value.nil?
127              value = ""
128            else
129              value = I18n.l value
130            end
131
132            ret += f.text_field(field.app_field_name, {:id => datepicker_id, :disabled
=> ('disabled' if field.visible != 2), :value => value})
133           end
134         when :time:
135           ret +=  "<p>#{label}"
136           token = rand(36**8).to_s(36)
137           timepicker_id = "#{card.id}_timepicker_#{field.app_field_name}_#{token}"
138           value = eval("f.object.#{field.app_field_name}")
139           ret += f.text_field(field.app_field_name, {:id => timepicker_id, :disabled
=> ('disabled' if field.visible != 2), :dojoType=>"dijit.form.TimeTextBox", :value => value, :lang
=> "de", :style => "width: 50px; margin-left: 10px"})
140           ret += "\n#{info}</p>"
141         when :datetime:
142           ret += "<p>#{label}"
143           if field.visible != 2
144             value = eval("l(field.#{field.app_field_name})")
145             ret += f.text_field(field.app_field_name, {:disabled => 'disabled', :value =>
value})
146           else
147             token = rand(36**8).to_s(36)
148             datepicker_id = "#{card.id}_datepicker_#{field.app_field_name}_#{token}"
149             timepicker_id = "#{card.id}_timepicker_#{field.app_field_name}_#{token}"
150
151             ret += f.text_field "#{field.app_field_name}date", {:id => datepicker_id,
:disabled => ('disabled' if field.visible != 2), :style => "width: 75px;"}
152             ret += f.text_field "#{field.app_field_name}time", {:id => timepicker_id,
:disabled => ('disabled' if field.visible != 2), :dojoType=>"dijit.form.TimeTextBox", :constraints
=> "{timePattern:'HH:mm'}" , :style => "width: 50px; margin-left: 10px"}
153             ret += "Uhr"
154           end
155           ret += "\n#{info}</p>"
156
157         when :text:
158           ret += "<p>#{label}#{f.text_area field.app_field_name, :onchange =>
"$j(this).form_element__worked(#{card.id})", :disabled => ('disabled' if field.visible != 2), :cols =>
35, :rows => 4}\n#{info}</p>"
159
160         when :password:
```

```
161         app_field_name = 'password' # Passwortfelder werden ohne dem Namensprefix
'crypted' angezeigt
162         ret += "<p>#{label}#{f.password_field app_field_name, :onchange =>
"$j(this).form_element__worked(#{card.id})", :disabled => ('disabled' if field.visible !=
2)}\n#{info}</p>"
163         ret += "<p><label>#{card.app_model.human_attribute_name
field.app_field_name}*</label>#{f.password_field app_field_name + '_confirmation', :onchange =>
"$j(this).form_element__worked(#{card.id})", :disabled => ('disabled' if field.visible !=
2)}\n#{info}</p>"
164
165       when :select_array:
166         ret += "<p>#{label}"
167         if @bpi.card[card.id].respond_to?"#{field.app_field_name}_id"
168           app_field_name = "#{field.app_field_name}_id"
169         else
170           app_field_name = field.app_field_name
171         end
172
173         token = rand(36**8).to_s(36)
174         select_id = "dselect_#{card.id}_#{field.app_field_name}_#{token}"
175
176         if field.card.app_model.respond_to?("#{field.app_field_name}_array")
177           # Klassenmethode 'irgendwas_array' aufrufen
178           ret += f.select app_field_name, ev-
al("@bpi.card[card.id].class.#{field.app_field_name}_array(card, @bpi)"), {}, {:id => select_id,
:disabled => ('disabled' if field.visible != 2), :dojoType => "dijit.form.FilteringSelect", :style =>
"margin: 0px; height: 16px; background: url(images/bg_form_text.gif) top left repeat-x; border: 1px solid #002b42; width:
286px;", :onchange => "$j(this).dijitFilteringSelect_worked(#{card.id})"}
179         else
180           # Instanzmethode 'irgendwas_array' aufrufen
181           ret += f.select app_field_name, ev-
al("@bpi.card[card.id].#{field.app_field_name}_array(card, @bpi)"), {}, {:id => select_id, :disabled
=> ('disabled' if field.visible != 2), :dojoType => "dijit.form.FilteringSelect", :style => "margin: 0px;
height: 16px; background: url(images/bg_form_text.gif) top left repeat-x; border: 1px solid #002b42; width: 286px;",
:onchange => "$j(this).dijitFilteringSelect_worked(#{card.id})"}
182         end
183         ret += "\n#{info}</p>"
184
185       when :select_belongs:
186         belongs_class =
@bpi.card[card.id].reflect_on_association_class_name(field.app_field_name.gsub(/_id$j/,
"")).constantize
187         label = belongs_class.human_attribute_name
field.app_field_name.gsub(/_id$j/, "")
188
189         value = eval("f.object.#{field.app_field_name}")
190
191         select_options = eval("#{belongs_class.to_s}.find :all")
192         select_options.unshift(eval("#{belongs_class.to_s}.new"))
193
194         token = rand(36**8).to_s(36)
195         select_id = "dselect_#{card.id}_#{field.app_field_name}_#{token}"
196
197         "<p><label for=\"#{select_id}\">#{label}</label>"
198         ret += f.select field.app_field_name, select_options.collect{ |i| [i.name,
i.id] }, {:selected => value }, {:id => select_id, :dojoType => "dijit.form.FilteringSelect",
:onchange => "$j(this).dijitFilteringSelect_worked(#{card.id})"}
199         ret += "\n#{info}</p>"
200
201       when :search_belongs:
202         # Bsp. wg. Parameter:
203         # <%= suchfeld @thing, @thing.owner, Owner, 'Besitzer', 'thing_owner_id',
'things' -%>
204         belongs_class =
@bpi.card[card.id].reflect_on_association_class_name(field.app_field_name).constantize
205
206         foreign_key_field =
@bpi.card[card.id].class.reflect_on_association(field.app_field_name.to_sym).options[:fo
reign_key]
207         foreign_key_field = "#{field.app_field_name}_id" if foreign_key_field.blank?
208
209         current_value_obj = eval("@bpi.card[card.id].#{field.app_field_name}")
210         current_value_id = current_value_obj.blank? ? nil : current_value_obj.id
```

```
211
212         select_options = eval("#{belongs_class.to_s}.find :all")
213         select_options.unshift(eval("#{belongs_class.to_s}.new"))
214
215         label = belongs_class.human_attribute_name field.app_field_name
216         token = rand(36**8).to_s(36)
217         select_id = "dselect_#{card.id}_#{field.app_field_name}_#{token}"
218
219         ret += "<p><label for=\"#{select_id}\">#{label}</label>"
220         ret += f.select foreign_key_field, select_options.collect{ |i| [i.label,
i.id] }, {:selected => current_value_id }, {:id => select_id, :disabled => ('disabled' if
field.visible != 2), :dojoType => "dijit.form.FilteringSelect", :style => "margin: 0px; height: 16px; back-
ground: url(images/bg_form_text.gif) top left repeat-x; border: 1px solid #002b42; width: 286px;", :onchange =>
"$j(this).dijitFilteringSelect_worked(#{card.id})"}
221         ret += "\n#{info}</p>"
222     end
223   end
224   return ret
225 end
226
227 # Helper zur Ausgabe einer Liste einer Card
228 def draw_card_list card
229   card_width_px = get_card_width_px card.width
230   @list_width_px = get_list_should_width_px card_width_px
231   unless @bpi.collection[card.id]==" "
232     @bpi.card[card.id] = paginate_finder params, card,
parse_conditions(card.special_conditions)
233   else
234     @bpi.card[card.id] = paginate_finder params, card,
parse_conditions(card.special_conditions)
235   end
236   return draw_card_list_for_collection card
237 end
238
239 # Helper zur Ausgabe einer Liste ohne @bpi.card zu überschreiben
240 def draw_card_list_for_collection card
241   ret = draw_card_list_head card.card_columns
242    # Zeile für das Neuanlegen eines Datensatzes
243   ret += "<div id=\"card_#{card.id}_new\" class=\"card_#{card.id}_row_0\" style=\"display:none\">
</div>"
244   row_number = 1
245   @bpi.card[card.id].each do |row|
246     ret << draw_card_list_row(card.card_columns, row, row_number, card.id)
247     row_number += 1
248   end
249   ret << draw_card_list_foot(card)
250   return ret
251 end
252
253 # Helper zur Ausgabe des Kopfes einer Card-Liste
254 def draw_card_list_head card_columns
255   ret = "<div class=\"tr_top\" style=\"width:#{@list_width_px}px\">"
256   ret << "<div class=\"th_left\"> </div>"
257   ret << "<div class=\"th_wp2_lever\"> </div>"
258
259   card_columns.each do |card_column|
260     column_type = card_column.app_field_type(@bpi.card[card_column.card.id]).to_s
261
262     unless card_column.app_field_name.include? 'label'
263         if card_column.association_name.blank?
264           label = card_column.card.app_model.human_attribute_name
card_column.app_field_name
265         else
266           belongs_class =
card_column.card.app_model.reflect_on_association_class_name(card_column.association_nam
e).constantize
267           label = belongs_class.human_attribute_name card_column.app_field_name
268         end
269         sort_image = sort_td_class_helper card_column.app_field_name, nil
270         field_content = sort_link_helper(label, card_column.app_field_name,
card_column.card, card_column.association_name)
271     else
272       if card_column.association_name.blank?
273         label = card_column.card.app_model.human_name
```

```
274         sort_image = sort_td_class_helper card_column.card.app_model_name.downcase,
nil
275         field_content = sort_link_helper(label,
card_column.card.app_model_name.downcase, card_column.card,
card_column.association_name)
276       else
277         belongs_class =
card_column.card.app_model.reflect_on_association_class_name(card_column.association_nam
e).constantize
278         label = belongs_class.human_name
279         sort_image = sort_td_class_helper card_column.app_field_name,
card_column.association_name.downcase
280         field_content = sort_link_helper(label, card_column.app_field_name,
card_column.card, card_column.association_name)
281       end
282     end
283
284     ret << "<div class=\"th head_#{column_type}\"
style=\"width:#{card_column.width}px\">#{sort_image} #{field_content}</div>"
285     ret << "<div class=\"th_hyphen\" style=\"width: #{LIST_HYPHEN_WIDTH}px\"> </div>"
286   end
287   ret << "<div class=\"th_working_icons\"> </div>"
288   ret << "<div class=\"th_right\"> </div>"
289   ret << "<div class=\"cleaner\"> </div>"
290   return ret + "</div>"
291 end
292
293 def draw_card_list_row card_columns, row, row_number, card_id
294   ret = "<div id=\"card_#{card_id}_#{row.class.to_s.downcase}_#{row.id}\"
class=\"card_#{card_id}_row_#{row_number}\" style=\"width:#{@list_width_px}px\" onmouseov-
er=\"focus_row(#{card_id}, #{row_number})\">"
295   ret << draw_card_list_row_content(card_columns, row)
296   ret + "</div>"
297 end
298
299 def draw_card_list_foot card
300   ret = ""
301   card_height_px = get_card_height_px card.height
302   # Höhe (Pixel) die innerhalb der Tabelle genutzt werden kann
303   list_height_standby_px = get_list_height_standby_px card_height_px
304   # Anzahl der Zeilen, die maximal in die Tabelle passen würden
305   list_row_count_max = get_list_row_count_max list_height_standby_px
306   rows_missing_count  = list_row_count_max - @bpi.card[card.id].length
307   1.upto(rows_missing_count) {|i|
308     row_number = @bpi.card[card.id].length + i
309     ret << "<div style=\"width:#{@list_width_px}px; height:#{LIST_ROW_HEIGHT}px\"
class=\"card_#{card.id}_row_#{row_number}\">"
310     ret << "<div class=\"td_left\" style=\"height:#{LIST_ROW_HEIGHT}px\"> </div>"
311     ret << "<div class=\"td wp2_lever\" style=\"height:#{LIST_ROW_HEIGHT}px; line-
height:#{LIST_ROW_HEIGHT}px\"> </div>"
312     card.card_columns.each do |card_column|
313       ret << "<div class=\"td\" style=\"width:#{card_column.width}px; height:#{LIST_ROW_HEIGHT}px;
line-height:#{LIST_ROW_HEIGHT}px\"> </div>"
314       ret << "<div class=\"td_hyphen\" style=\"width: #{LIST_HYPHEN_WIDTH}px;
height:#{LIST_ROW_HEIGHT}px\"> </div>"
315     end
316     ret << "<div class=\"td working_icons\" style=\"height:#{LIST_ROW_HEIGHT}px; line-
height:#{LIST_ROW_HEIGHT}px\"> </div>"
317     ret << "<div class=\"td_right\" style=\"height:#{LIST_ROW_HEIGHT}px\"> </div>"
318     ret << "<div class=\"cleaner\"> </div>"
319     ret << "</div>"
320   }
321   ret << "<div class=\"tr_bottom\" style=\"width:#{@list_width_px}px\">"
322   ret << "<div class=\"td_bottom_left\"> </div>"
323   ret << "<div class=\"td_bottom_wp2_lever\"> </div>"
324   card.card_columns.each { |column|
325     ret << "<div class=\"td_bottom\" style=\"width:#{column.width}px;\"> </div>"
326     ret << "<div class=\"td_bottom_hyphen\" style=\"width: #{LIST_HYPHEN_WIDTH}px\"> </div>"
327   }
328   ret << "<div class=\"td_bottom_working_icons\"> </div>"
329   ret << "<div class=\"td_bottom_right\" > </div>"
330   ret << "<div class=\"cleaner\"> </div>"
331   # div für die Listeninformation
```

```
332   ret << "<div id=\"card_#{card.id}_list_info\"
class=\"#{list_row_count_max}_#{LIST_ROW_HEIGHT}_#{card.edit_level_2_lines}\" style=\"display:
none\"> </div>"
333   ret << "</div>"
334 end
335
336 def render_card_item card_item, object, f, card, wp_width_min
337   ret = {"content" => "", "label" => ""}
338   column_type = card_item.app_field_type(object)
339   ret["label"] = card.app_model.human_attribute_name card_item.app_field_name
340   case column_type
341     when :string:
342       width = card_item.width >= wp_width_min[:input_character] ? card_item.width :
wp_width_min[:input_character]
343       width -= 3
344       if card_item.association_name.blank?
345         ret["content"] = f.text_field card_item.app_field_name, :onchange =>
"$j(this).form_element_worked(#{card.id});", :disabled => ('disabled' if card_item.visible != 2),
:style => "width: #{width}px"
346       else
347         field_name = card_item.app_field_name.blank? ? 'label' :
card_item.app_field_name
348         assoc = eval("object.#{card_item.association_name}")
349         assoc_2level = eval("assoc.#{field_name}") unless assoc.blank?
350         if !assoc_2level.blank? && assoc_2level.class.methods.include?('base_class') &&
assoc_2level.class.base_class.descends_from_active_record?
351           assoc = assoc_2level
352           field_name = 'label'
353         end
354         ret["content"] = eval("assoc.#{field_name}") unless assoc.blank?
355       end
356     when :integer
357       width = card_item.width >= wp_width_min[:input_numeric] ? card_item.width :
wp_width_min[:input_numeric]
358       width -= 3
359       ret["content"] = f.text_field card_item.app_field_name, :onchange =>
"$j(this).form_element_worked(#{card.id});", :disabled => ('disabled' if card_item.visible != 2),
:style => "width: #{width}px"
360     when :decimal, :float:
361       width = card_item.width >= wp_width_min[:input_numeric] ? card_item.width :
wp_width_min[:input_numeric]
362       width -= 3
363       ret["content"] = f.amount_field card_item.app_field_name, :onchange =>
"$j(this).form_element_worked(#{card.id});", :disabled => ('disabled' if card_item.visible != 2),
:style => "width: #{width}px"
364     when :boolean:
365       ret["content"] = f.check_box card_item.app_field_name, :onchange =>
"$j(this).form_element_worked(#{card.id});", :disabled => ('disabled' if card_item.visible != 2)
366     when :date:
367       if card_item.visible != 2
368         # TODO: Date korrekt auch in Feldern anzeigen
369         ret["content"] = f.text_field card_item.app_field_name, :onchange =>
"$j(this).form_element_worked(#{card.id});", :disabled => 'disabled', :style => "width: #{width}px"
370       else
371        token = rand(36**8).to_s(36)
372        datepicker_id = "#{card.id}_datepicker_#{card_item.app_field_name}_#{token}"
373        value = eval("f.object.#{card_item.app_field_name}")
374        if value.nil?
375          value = ""
376        else
377          value = I18n.l value
378        end
379        ret["content"] = f.text_field(card_item.app_field_name, {:id => datepicker_id,
:value => value, :onchange => "$j(this).form_element_worked(#{card.id});", :style => "width:
#{wp_width_min[:input_date]}px"})
380       end
381     when :time:
382       token = rand(36**8).to_s(36)
383       timepicker_id = "#{card.id}_timepicker_#{card_item.app_field_name}_#{token}"
384       value = eval("f.object.#{card_item.app_field_name}")
385       ret["content"] = f.text_field(card_item.app_field_name, {:id => timepicker_id,
:onchange => "$j(this).form_element_worked(#{card.id});", :dojoType=>"dijit.form.TimeTextBox", :value =>
value, :lang => "de", :style => "#{INPUT_TIME_STYLE} width: #{wp_width_min[:input_time]}px"})
```

```
386     when :datetime:
387       if card_item.visible != 2
388         value = eval("l(field.#{card_item.app_field_name})")
389         ret += f.text_field(card_item.app_field_name, {:disabled => 'disabled', :value
=> value})
390       else
391         token = rand(36**8).to_s(36)
392         datepicker_id = "#{card.id}_datepicker_#{card_item.app_field_name}_#{token}"
393         timepicker_id = "#{card.id}_timepicker_#{card_item.app_field_name}_#{token}"
394         ret["content"] = f.text_field "#{card_item.app_field_name}date", {:id => date-
picker_id, :style => "width: #{wp_width_min[:input_date]}px"}
395         ret["content"] += f.text_field "#{card_item.app_field_name}time", {:id => time-
picker_id, :dojoType=>"dijit.form.TimeTextBox", :constraints => "{timePattern:'HH:mm'}" , :style =>
"#{INPUT_TIME_STYLE} width: #{wp_width_min[:input_time]}px"}
396         ret["content"] += "Uhr"
397       end
398     when :text:
399       width = card_item.width >= wp_width_min[:input_character] ? card_item.width :
wp_width_min[:input_character]
400       width -= 3
401       ret["content"] = f.text_area card_item.app_field_name, :onchange =>
"$j(this).form_element_worked(#{card.id});", :disabled => ('disabled' if card_item.visible != 2),
:cols => 35, :rows => 4, :style => "width: #{width}px"       # Sollen Passwörter in Bearbei-
tungsebene 1 mit angezeigt werden?
402     when :password:
403       width = card_item.width >= wp_width_min[:input_character] ? card_item.width :
wp_width_min[:input_character]
404       width -= 3
405       app_field_name = 'password' # Passwortfelder werden ohne dem Namensprefix
'crypted' angezeigt
406       ret["content"] = f.password_field app_field_name, :onchange =>
"$j(this).form_element_worked(#{card.id});", :disabled => ('disabled' if card_item.visible != 2),
:style => "width: #{width}px"
407       ret["content"] += "<br/>"
408       ret["content"] += f.password_field app_field_name + '_confirmation', :onchange =>
"$j(this).form_element_worked(#{card.id});", :disabled => ('disabled' if card_item.visible != 2),
:style => "width: #{width}px"
409     when :select_array:
410       width = card_item.width >= wp_width_min[:combobox] ? card_item.width :
wp_width_min[:combobox]
411       width -= 3
412       if @bpi.card[card.id].respond_to?"#{card_item.app_field_name}_id"
413         app_field_name = "#{card_item.app_field_name}_id"
414       else
415         app_field_name = card_item.app_field_name
416       end
417       token = rand(36**8).to_s(36)
418       select_id = "dselect_#{card.id}_#{card_item.app_field_name}_#{token}"
419       if card_item.card.app_model.respond_to?("#{card_item.app_field_name}_array")
420         ret["content"] = f.select app_field_name, ev-
al("@bpi.card[card.id].class.#{card_item.app_field_name}_array(card, @bpi)"), {}, {:id => select_id,
:onchange => "$j(this).dijitFilteringSelect_worked(#{card.id})", :disabled => ('disabled' if
card_item.visible != 2), :dojoType => "dijit.form.FilteringSelect", :style => "#{COMBOBOX_STYLE}
width: #{width}px;"}
421       else
422         # Instanzmethode 'irgendwas_array' aufrufen
423         ret["content"] = f.select app_field_name, ev-
al("object.#{card_item.app_field_name}_array(card, @bpi)"), {}, {:id => select_id, :onchange =>
"$j(this).dijitFilteringSelect_worked(#{card.id})", :disabled => ('disabled' if card_item.visible != 2),
:dojoType => "dijit.form.FilteringSelect", :style => "#{COMBOBOX_STYLE} width: #{width}px;"}
424       end
425     when :select_belongs:
426       width = card_item.width >= wp_width_min[:combobox] ? card_item.width :
wp_width_min[:combobox]
427       width -= 3
428       belongs_class =
@bpi.card[card.id].reflect_on_association_class_name(card_item.app_field_name.gsub(/_id$
/, "")).constantize
429       current_value_obj = eval("@bpi.card[card.id].#{card_item.app_field_name}")
430       current_value_id = current_value_obj.id unless current_value_obj.blank?
431       select_options = eval("#{belongs_class.to_s}.find :all")
432       select_options.unshift(eval("#{belongs_class.to_s}.new"))
433       token = rand(36**8).to_s(36)
```

```
434         select_id = "dselect_#{card.id}_#{card_item.app_field_name}_#{token}"
435         ret["content"] = f.select card_item.app_field_name, select_options.collect{ |i|
[i.name, i.id] }, {:selected => current_value_id }, {:id => select_id, :onchange =>
"$j(this).dijitFilteringSelect_worked(#{card.id})", :disabled => ('disabled' if card_item.visible != 2),
:dojoType => "dijit.form.FilteringSelect", :style => "#{COMBOBOX_STYLE} width: #{width}px;"}
436       when :search_belongs:
437         width = card_item.width >= wp_width_min[:combobox] ? card_item.width :
wp_width_min[:combobox]
438         width -= 3
439         belongs_class =
@bpi.card[card.id].reflect_on_association_class_name(card_item.app_field_name).constanti
ze
440         foreign_key_field =
@bpi.card[card.id].class.reflect_on_association(card_item.app_field_name.to_sym).options
[:foreign_key]
441         foreign_key_field = "#{card_item.app_field_name}_id" if foreign_key_field.blank?
442         current_value_obj = eval("object.#{card_item.app_field_name}")
443         current_value_id = current_value_obj.blank? ? nil : current_value_obj.id
444         select_options = belongs_class.all
445         select_options.unshift(belongs_class.new)
446         token = rand(36**8).to_s(36)
447         select_id = "dselect_#{card.id}_#{card_item.app_field_name}_#{token}"
448         ret["content"] = f.select foreign_key_field, select_options.collect{ |i|
[i.label, i.id] }, {:selected => current_value_id }, {:id => select_id,:onchange =>
"$j(this).dijitFilteringSelect_worked(#{card.id})", :disabled => ('disabled' if card_item.visible != 2),
:dojoType => "dijit.form.FilteringSelect", :style => "#{COMBOBOX_STYLE} width: #{width}px;"}
449       end
450     return ret
451   end
452
453   # berechne die Breite (Pixel) einer Karte
454   def get_card_width_px card_width
455     (card_width * CARD_WIDTH + (card_width - 1) * CARD_FREEBOARD)
456   end
457
458   # berechne die Höhe (Pixel) einer Karte
459   def get_card_height_px card_height
460     card_height * CARD_HEIGHT
461   end
462
463   # berechne die Breite (Pixel) einer Liste mit den aktuellen Spalten
464   def get_list_width_px columns
465     card_list_width_px = LIST_WIDTH_RESERVED
466     columns.each do |card_column|
467       card_list_width_px += card_column.width + LIST_HYPHEN_WIDTH
468     end
469     return card_list_width_px
470   end
471
472   # berechne die SOLL-Breite (Pixel) einer Liste
473   def get_list_should_width_px card_width_px
474     card_width_px - CARD_BORDER_WIDTH * 2 - LIST_COMPRESSION_HORIZONTAL
475   end
476
477   # berechne die Höhe (Pixel) einer Liste
478   def get_list_height_px card_height_px
479     card_height_px - LIST_HEIGHT_RESERVED
480   end
481
482   # berechne die Breite (Pixel) einer Cardtitelleiste
483   def get_card_headbar_width_px card_width_px
484     card_width_px - CARD_BORDER_WIDTH * 2 - CARD_HEADBAR_BORDER_WIDTH * 2
485   end
486
487   # berechne die Breite (Pixel) des unteren Cardrandes
488   def get_card_border_bottom_width_px card_width_px
489     card_width_px - CARD_BORDER_WIDTH * 2
490   end
491
492   # berechne die Breite (Pixel) einer Bearbeitungsebene 2
493   def get_wp2_width_px card_width_px
494     card_table_width_px = (card_width_px - CARD_BORDER_WIDTH * 2) -
LIST_COMPRESSION_HORIZONTAL
495     # - td_left + card_table_width_px
496     card_table_width_px  - LIST_BORDER_WIDTH * 2
497   end
```

```
498
499 # berechne die Höhe (Pixel) einer Bearbeitungsebene 2
500 def get_wp2_height_px wp2_rows_count
501   wp2_rows_count * LIST_ROW_HEIGHT
502 end
503
504 # berechne die Höhe (Pixel) die in einer Liste zur Verfügung steht
505 def get_list_height_standby_px card_height_px
506   card_height_px - CARD_HEIGHT_RESERVED - LIST_HEIGHT_RESERVED
507 end
508
509 # berechne die Anzahl der Zeilen, die maximal in die Tabelle passen würden
510 def get_list_row_count_max list_height_standby_px
511   list_height_standby_px / LIST_ROW_HEIGHT
512 end
513
514 def close_wp
515   begin
516     # TODO: Das Neuanlegen von Datensätzen in der Liste wird derzeit nur in eigenen
CardPartials/RowPartials unterstützt, die darauf vorbereitet sind, da für das Abbrechen
in der Standard-Card in dieser Methode noch ein data_object benötigt wird.
517     data_object_id = params[:id].split("_")[0].to_i
518     data_object = params[:controller].classify.constantize.find(data_object_id) un-
less data_object_id == 0
519     card = Card.find(params[:id].split("_")[1].to_i)
520
521     render :update do |page|
522       if card.cardpartial.blank?
523         # old prototype syntax: page <<
"$j('#card#{card.id}').style.height='#{card.max_height}px';" unless
card.max_height.blank?
524         #page << "$j('#card#{card.id}').css('height','#{card.max_height}px');" un-
less card.max_height.blank?
525         row_content = draw_card_list_row_content(card.card_columns, data_object)
526         row_id = "card_#{card.id}_#{data_object.class.to_s.downcase}_#{data_object.id}"
527         page.call("close_wp_and_reload_row", "#{row_id}", "#{row_content}", "#{card.id}")
528         #page.replace_html
"card_#{card.id}_#{data_object.class.to_s.downcase}_#{data_object.id}",
draw_card_list_row_content(card.card_columns,data_object)
529         page << "card_saved(#{card.id})";
530       else
531         page.replace_html "card_#{card.id}", :partial => "cardviews/#{card.cardpartial}",
:locals => {:card => card}
532       end
533     end
534   rescue Exception => ex
535     @bpi.card[card.id].errors.add(card.app_model_name, ex)
536     report_errors(card, @bpi.card[card.id])
537     logger.info "error on close_wp: #{ex}"
538     return
539   end
540 end
541
542 def create_from_wp
543   begin
544     card = Card.find(params[:card_id])
545     data_object =
card.app_model.new(params[card.app_model_name.tableize.singularize.to_sym])
546     @bpi.card[card.id] = data_object
547
548     # try to validate/save the card
549     assign_with_parent card
550     if @bpi.card[card.id].save
551       flash[:notice] = 'Erfolgreich angelegt!'
552       render_card card
553     else
554       # otherwise report errors
555       report_errors(card, data_object)
556     end
557   rescue Exception => ex
558     @bpi.card[card.id].errors.add(card.app_model_name, ex)
559     report_errors(card, @bpi.card[card.id])
560     logger.info "error on create_from_wp: #{ex}"
561     return
562   end
563 end
```

```
564
565 def update_from_wp
566   data_object = pa-
rams[:controller].classify.constantize.find(params[:id].split("_")[0].to_i)
567   card = Card.find(params[:id].split("_")[1].to_i)
568   # Daten validiert und gespeichert?
569   if (da-
ta_object.update_attributes(params[data_object.class.to_s.tableize.singularize.to_sym]))
570     render :update do |page|
571       # Karte als fertig bearbeitet markieren
572       if card.cardpartial.blank?
573         row_id = "card_#{card.id}_#{data_object.class.to_s.downcase}_#{data_object.id}"
574         row_content = draw_card_list_row_content(card.card_columns, data_object)
575         page.call("close_wp_and_reload_row", "#{row_id}", "#{row_content}", "#{card.id}")
576       else
577         page.replace_html "card_#{card.id}", :partial => "cardviews/#{card.cardpartial}",
:locals => {:card => card}
578       end
579       page << "card_saved(#{card.id});"
580     end
581     # Validierung fehlgeschlagen
582   else
583     report_errors(card, data_object)
584   end
585 end
586
587 def load_working_planes_for_create
588   card = Card.find(params[:id])
589   @bpi.card[card.id] = card.app_model.new
590   set_current_user card, current_user
591   assign_with_parent card
592   @bpi.card[card.id].before_load @bpi
593   wp2_lever_image = "icons/wp2_closed.png"
594   wp2_lever_title = "aufklappen"
595   render :update do |page|
596     page.replace_html "card_#{card.id}_new", :partial => "card_forms/working_planes_create",
:locals => {:card => card, :object => @bpi.card[card.id], :wp2_lever_image =>
wp2_lever_image, :wp2_lever_title => wp2_lever_title}
597     page << "$j(\"#card_#{card.id}_new\").css(\"display\", \"inline\");"
598     page << "recentlyRenderedCards.push(#{card.id})"
599     page << "card_has_unsaved_data(#{card.id});"
600     page.call("wp")
601   end
602 end
603
604 def load_working_planes_for_update
605   card = Card.find(params[:card_id])
606   object = params[:object].constantize.find(params[:id])
607   wp2_lever_image = "icons/wp2_closed.png"
608   wp2_lever_title = "aufklappen"
609
610   if params[:open_wp2]
611     wp2_lever_image = "icons/wp2_open.png"
612     wp2_lever_title = "zuklappen"
613   end
614   render :update do |page|
615     page.replace_html "card_#{card.id}_#{object.class.to_s.downcase}_#{object.id}",
:partial => 'card_forms/working_planes_update', :locals => {:card => card, :object => object,
:wp2_lever_image => wp2_lever_image, :wp2_lever_title => wp2_lever_title}
616     page << "recentlyRenderedCards.push(#{params[:card_id]});"
617     page.call("wp")
618     if params[:open_wp2]
619       page << "open_wp2(#{card.id},
'card_#{card.id}_#{object.class.to_s.downcase}_#{object.id}');"
620     end
621   end
622 end
```

Anhang E - Quelltexte (JavaScript/jQuery)

```
// speichert alle Card_id's, die mit dem letzten Ajax-Aufruf gerendert wurden
var recentlyRenderedCards = new Array();

function closeIt()
{
  if($(".unsaved").length)
  {
    return "Sie haben ungesicherte Daten.\nDie Daten gehen verloren, wenn Sie fortfahren."
  }
}

window.onbeforeunload = closeIt;

/*
 * - führt das nötige JavaScript zu einer Card aus
 *  - speichert die belongs_to -inputs (dijit.form.FilteringSelect)
 *   - durch die gespeicherte Variable kann die dijit.form.FilteringSelect wieder zerstört werden, wenn die Card neugeladen wird
 *  - generiert Date- und Timepicker
*/

function combobox_should_fire_onchange(evt)
{
  combobox_trigger = evt.target;
  var id_split =  combobox_trigger.id.split("_");
  id_split.shift();
  var card_id = id_split.shift();
  id_split.pop()
  var field_name = id_split.join("_");
  eval(field_name + "_" + card_id + ".attr('value','"+ $("#"+combobox_trigger.id).val()+"')");
}

function make_necessary_javaScript(card_id)
{
  // für jede dijit.form.FilteringSelect
  $("input[id^='dselect_" + card_id + "']").each(function()
  {
    var id_split = $(this).attr("id").split("_");
    id_split.shift();
    var card_id = id_split.shift();
    id_split.pop()
    var field_name = id_split.join("_");

    // wurde für diese Card schonmal diese dijit.form.FilteringSelect erstellt?
    if(eval("typeof " + field_name + "_" + card_id + "!= 'undefined'"))
    {
      // zerstöre die bisher gespeicherte dijit.form.FilteringSelect
      eval(field_name + "_" + card_id + ".destroy()");
    }
    // speicher die dijit.form.FilteringSelect
    eval(field_name + "_" + card_id + "= dijit.byId($(this).attr('id'))");
    //dojo.connect(eval(field_name + "_" + card_id), 'onKeyUp', "combo-box_should_fire_onchange");
  }
  );

  // für jede dijit.form.TimeTextBox
  $("input[id^='" + card_id + "_timepicker']").each(function()
  {
    var id_split = $(this).attr("id").split("_");
    id_split.shift();
    var card_id = id_split.shift();
    id_split.pop()
    var field_name = id_split.join("_");

    // wurde für diese Card schonmal diese dijit.form.TimeTextBox erstellt?
    if(eval("typeof tp_" + card_id + "_" + field_name + " != 'undefined'"))
    {
      // zerstöre die bisher gespeicherte dijit.form.TimeTextBox
      eval("tp_" + card_id + "_" + field_name + ".destroy()");
    }
    // speicher die dijit.form.TimeTextBox
```

```
 71      eval("tp_" + card_id + "_" + field_name + "= dijit.byId($(this).attr('id'))");
 72      //$(this).change(function(){merge_date_and_time(card_id,
field_name)}).focus(function(){merge_date_and_time(card_id,
field_name)}).blur(function(){merge_date_and_time(card_id, field_name)});
 73    });
 74
 75    // für jeden datepicker
 76    $("input[id^='" + card_id + "_datepicker']").each(function()
 77    {
 78      eval("datePickerController.createDatePicker({ formElements:{\"" + $(this).attr("id") + "\":\"" +
datepicker_format + "\"},\"lang\":\"" + datepicker_lang + "\", showWeeks:true, statusFormat:\"l-cc-sp-d-
sp-F-sp-Y\", fillGrid:true, constrainSelection:false, noFadeEffect:true});");
 79    });
 80
 81    // für jedes Element der Bearbeitungsebene 1 (wp1 = working plane 1 = Arbeitsebene
1)
 82    $("div[id^='wp1_column_']").each(function()
 83    {
 84      $(this).find("input").focus(function()
 85      {
 86        $(".wp1_column_relieved").removeClass("wp1_column_relieved");
 87        $(this).closest("div[id^='wp1_column_']").addClass("wp1_column_relieved");
 88      })
 89      .blur(function()
 90      {
 91        $(".wp1_column_relieved").removeClass("wp1_column_relieved");
 92      }
 93      );
 94    });
 95  }
 96
 97  function card_has_unsaved_data(card_id)
 98  {
 99    $("#card_" + card_id + "__headline").css("color","#ffc840");
100    $("#card_" + card_id).addClass("unsaved");
101  }
102
103  // Card wurde gespeichert
104  function card_saved(card_id)
105  {
106    $("#card_" + card_id).removeClass("unsaved");
107    $("#card_" + card_id).find(".card_headbar").css("color", "");
108    $("#card_" + card_id + "__validation_errors").html("");
109    $("#card_" + card_id + "__validation_errors").css("display", "none");
110  }
111
112  function has_errors(card_id, type, field)
113  {
114    if($("input[id='" + type + "_" + field + "']").length > 0)
115    {
116      $("input[id='" + type + "_" + field + "']").addClass("has_errors");
117    }
118    // eine dijit.FilteringSelect hat den Fehler verursacht
119    else
120    {
121      var field_split = field.split("_");
122      $("div[id^='widget_dselect_" + card_id + "_" + field_split[0] + "']").css("border","1px solid
#ff0000");
123    }
124  }
125
126  // Fokussieren einer Zeile
127  function focus_row(card_id, row_number)
128  {
129    $(".focus").removeClass("focus");
130    $(".card_" + card_id + "_row_" + row_number).addClass("focus");
131    // zeile.spalten.zweite_spalte.finde_einen_link.fokussieren()
132    $("tr.card_" + card_id + "_row_" + row_number).children().eq(1).find("a").focus();
133  }
134
135
136
137  // Cards zurechtrücken; für eventuell auftretene vergrößerte Cardinhalte
138  function adjust_cards()
```

```
{
  // Abstand nach oben; der Wert ist in config/initializers/constanst.rb unter HEAD_HEIGHT einzustellen
  var row_offset__top = head_height;

  var raster_rows = parseInt($("#process_body").attr("class")) + 1;

  // Reihe hat mindestens eine Karte?
  var row_has_cards;

  // Maximale Cardhöhe der Reihe
  var row_active__height_max;

  for (var row_active = 1; row_active < raster_rows; row_active++)
  {

    // TODO: Cards die höher als 1 sind werden falsch interpretiert; 350 muss abgelöst werden von der CARD_HEIGHT -Konstante
    // Standardhöh+e = 350 (px)
    row_active__height_max = 350;

    row_has_cards = false;

    $(".rowmarker_" + row_active).each(function()
    {
      if(this.offsetTop > row_active__height_max)
      {
        row_active__height_max = this.offsetTop;
      }
      // Reihe hat mindestens eine Card
      row_has_cards = true;
    })
    if(row_has_cards)
    {
       row_offset__top = row_offset__top + parseInt(row_active__height_max) + 5;
    }

    var row_next = row_active + 1;

    if(row_next <= raster_rows)
    {
      // Cards aus den darunterliegenden Reihen werden nach unten verschoben
      $(".row_" + row_next).css({ "top" : row_offset__top });
    }
  }

  // hat die aktuelle Seite eine Card?
  if($("div[class^='rowmarker_']").length > 0)
  {
    // Fußzeile nach unten verschieben
    $("#footer").css("position", "absolute");
    $("#footer").css("top", row_offset__top);
  }
   $("#footer").css("display", "block");
}

this.tooltip = function(){
  xOffset = 10;
  yOffset = 20;

  $(".td").hover(function(e){
    if(this.title.length)
    {
      this.t = this.title;
      this.title = "";
      $("body").append("<p id='tooltip'>"+ this.t +"</p>");
      $("#tooltip")
        .css("top",(e.pageY - xOffset) + "px")
        .css("left",(e.pageX + yOffset) + "px")
        .fadeIn("fast");
    }
    else
    {
      this.t = "";
    }
```

```
212     },
213     function(){
214       this.title = this.t;
215       $("#tooltip").remove();
216     });
217     $(".td").mousemove(function(e){
218       $("#tooltip")
219         .css("top",(e.pageY - xOffset) + "px")
220         .css("left",(e.pageX + yOffset) + "px");
221     });
222 };
223
224
225 // ist keine Bearbeitungsebene gerade aktiv?
226 function wp_active()
227 {
228   if(typeof wp1_submit != 'undefined')
229   {
230     return true;
231   }
232   else
233   {
234     return false;
235   }
236 }
237
238 // Zumachen der Bearbeitungsebene für einen neuen Datensatz
239 function wp_close_new(card_id)
240 {
241   $("#card_" + card_id + "_new").html("");
242   $("#card_" + card_id + "_list_new").css("display", "none");
243   wp1_submit = undefined;
244   wp2_close(card_id);
245   card_saved(card_id);
246 }
247
248 function wp()
249 {
250   // Ist eine Bearbeitungsebene aktiv?
251   if($("#wp1_column_0").length)
252   {
253    wp1_submit = $("input[src='images/icons/check.png']");
254   }
255 }
256
257 // Bei Betätigen des B2-Hebels
258 function wp2_lever_apply(trigger, card_id)
259 {
260   img = trigger.find("img");
261   console.debug(img);
262   if(trigger.attr("title") == "aufklappen")
263   {
264     trigger.attr("title", "zuklappen");
265     row_trigger_id = trigger.closest("div[class^='card_"+card_id +"_"+ "row']").attr("id");
266     open_wp2(card_id, row_trigger_id);
267     img.attr("src", "/images/icons/wp2_open.png");
268   }
269   else
270   {
271     trigger.attr("title", "aufklappen");
272     wp2_close(card_id);
273     img.attr("src", "/images/icons/wp2_closed.png");
274   }
275 }
276
277 // öffne die zweite Bearbeitungsebene
278 function open_wp2(card_id, row_trigger_id)
279 {
280   // rows_count: Anzahl der Zeilen, in der die B2 geladen werden soll
281   // row_trigger: Nummer der Zeile, zu der die B2 gehört
282   row_trigger = parseInt($("#" + row_trigger_id).attr("class").split("_")[3]);
283
284   list_info_split = $("#card_" + card_id + "_list_info").attr("class").split("_");
285   // Anzahl der Zeilen, die aktuell in der Liste sind
286   //console.log(list_info_split[0]);
```

```
287   list_rows_count = parseInt(list_info_split[0]);
288   // Anzahl der Zeilen, die maximal in die Lioste passt
289   // Nummer der ersten Reihe
290   list_row_height_px = parseInt(list_info_split[1]);
291
292   rows_count = parseInt(list_info_split[2]);
293
294   wp2_height_px = list_row_height_px * rows_count;
295
296   row_first = 1;
297   // Nummer der letzten Reihe
298   row_last = list_rows_count;
299   // Anzahl der Zeilen, die noch verschwinden müssen
300   rows_to_dispose = rows_count;
301
302   //console.log("Zeilen die noch verschwinden müssen:" + rows_to_dispose);
303
304   while(rows_to_dispose > 0)
305   {
306
307     //Abstand nach oben
308     distance_to_first_row = row_trigger - row_first;
309     // Abstand nach unten
310     distance_to_last_row = row_last - row_trigger;
311
312     //console.log("rows_to_dispose: " + rows_to_dispose);
313     //console.log("distance_to_first_row: " + distance_to_first_row);
314     //console.log("distance_to_last_row: " + distance_to_last_row);
315
316     if(distance_to_first_row > distance_to_last_row)
317     {
318      row_vanish = row_first
319      row_first ++;
320     }
321     else
322     {
323       row_vanish = row_last
324       row_last--;
325     }
326
327     if(row_vanish == row_trigger)
328     {
329       rows_to_dispose = 0;
330       console.log("Zu wenig Zeilen für die zweite Bearbeitungsebene vorhanden");
331     }
332     else
333     {
334       //console.log("row_vanish: " + row_vanish);
335       $(".card_" + card_id + "_row_" + row_vanish).addClass("vanish");
336       //console.log("#card_" + card_id + "_row_" + row_vanish);
337       rows_to_dispose--;
338     }
339   }
340   // Zeilen werden ausgeblendet
341   $(".vanish").animate(
342     { height: "0px" },
343     500,
344     "",
345     function(){ $(this).css("display", "none") }
346   );
347
348   // Einblenden der B2
349   $("#wp2").animate(
350     { height: wp2_height_px + "px" },
351     500
352   );
353 }
354
355 function wp2_close(card_id)
356 {
357   list_info_split = $("#card_" + card_id + "_list_info").attr("class").split("_");
358
359   list_row_height_px = parseInt(list_info_split[2]);
360
361   // Zeilen werden wieder eingeblendet
362   $(".vanish").css("display", "inline");
```

```
363
364   $(".vanish").animate(
365     { height: list_row_height_px + "px" },
366     500,
367     "",
368     function(){ $(this).removeClass("vanish"); }
369   );
370
371   // Ausblenden der B2
372   $("#wp2").animate(
373     { height: "0px" },
374     500
375   );
376 }
377
378 // ist gerade eine zweite Bearbeitungsebene offe?
379 function is_wp2_open()
380 {
381   if(parseInt($("#wp2").css("height")) > 0)
382   {
383     return true;
384   }
385   else
386   {
387     return false;
388   }
389 }
390
391 function close_wp_and_reload_row(row_id, content, card_id)
392 {
393   if(is_wp2_open())
394   {
395     wp2_close(card_id);
396     setTimeout(function() {$("#" + row_id).html(content);},500)
397   }
398   else
399   {
400     $("#" + row_id).html(content);
401   }
402 }
403
404
405 $(document).ready(function()
406 {
407   /*
408    *  Initialisierung
409    */
410   dojo.require("dijit.form.FilteringSelect");
411   dojo.require("dijit.form.ComboBox");
412   dojo.require("dijit.form.TimeTextBox");
413   dojo.require("dojo.parser");
414
415   // dojo.parser laufen lassen, damit Comboboxen gerendert werden
416   dojo.parser.parse();
417
418   tooltip();
419
420   /* Breite der Kopfleiste ausreichend groß machen */
421   // linker Kopfleistenrand + Kopfleistenlogo + Kopfleistenprozesse + rechter Kopf-
leistenrand + Sicherheitsabstand
422   headbar_width = 13 + $("#headbar_logo").width() + $("#headbar_processes").width() + 13 +
15
423   $("#headbar").css("width", headbar_width + "px");
424
425   // hat die aktuelle Seite eine Karte?
426   if($("div[class^='row_']").length)
427   {
428     // Cards zurechtrücken
429     adjust_cards();
430   }
431
432   while(recentlyRenderedCards.length > 0)
433   {
434     make_necessary_javaScript(recentlyRenderedCards.pop());
435   }
```

```
436
437   // wenn der Mauszeiger eine Card betritt
438   $(".card_body").mouseenter(function()
439   {
440     $(".focus").removeClass("focus");
441   });
442
443   $("#spinner").hide(500);
444
445   $(this).ajaxStart(function()
446   {
447     $("#spinner").show(500);
448   }
449   );
450
451   $(this).ajaxStop(function()
452   {
453     $("#spinner").hide(500);
454   }
455   );
456
457
458   // nach fertigen Ajaxaufrufen
459   $(this).ajaxComplete(function()
460   {
461     // dojo.parser laufen lassen, damit Comboboxen gerendert werden
462     dojo.parser.parse();
463      // Cards zurechtrücken
464     adjust_cards();
465
466     while(recentlyRenderedCards.length > 0)
467     {
468       make_necessary_javaScript(recentlyRenderedCards.pop());
469     }
470
471     // Loops through the current list of in-memory datePickers and checks to see if
their associated form element exists within the DOM; if not, the datePicker is itself
removed from the DOM (and browser memory).
472     datePickerController.cleanUp();
473
474   });
475
476   $(this).ajaxSend(function(event, XMLHttpRequest)
477   {
478     // wenn Bearbeitungsebene aktiv ist
479     if(wp_active())
480     {
481       // Form der Bearbeitungsebene abschicken
482       wp1_submit.click();
483       // ist die Bearbeitungsebene immer noch offen? (tritt ein bei Validierungs-
fehlern auf)
484       if(wp_active())
485       {
486         // Breche den aktuellen Request ab
487         XMLHttpRequest.abort();
488       }
489     }
490   }
491   );
492
493   // Links aus Tabellenreihen kriegen folgendes click-event zugewiesen
494   $("td a").livequery('click', function()
495   {
496     var tr_id = $(this).closest("div[id^='card_']").attr("id");
497     var tr_id_split = tr_id.split("_");
498     var card_id = tr_id_split[1];
499     $("div[class^='card_" + card_id + "_row']").removeClass("highlight");
500     $("#" + tr_id).addClass("highlight");
501   });
502
503   $.fn.dijitFilteringSelect_worked = function(card_id)
504   {
505     $("#widget_" + $(this).attr("id")).css("border", "1px solid #ffd700");
506     card_has_unsaved_data(card_id);
507   }
508
```

```
509   $.fn.form_element__worked = function(card_id)
510 {
511   $(this).addClass("worked");
512   card_has_unsaved_data(card_id);
513 }
514
515   /*
516   * Tastatureingaben
517   */
518
519   $(this).keydown(function(event)
520   {
521       //if (!event) event = window.event; // für den IE später
522       switch(event.keyCode)
523       {
524         // Pfeiltaste runter
525         case 40:
526           // Aufruf aus dem Suchfeld
527           if($(event.target).attr("class")=="search_text")
528           {
529             //scrollen unterbinden
530             event.preventDefault();
531             trigger_id_split = $(event.target).attr("id").split("_");
532             focus_row(trigger_id_split[1], 0);
533           }
534           // hat gerade eine Zeile den Fokus?
535           else if($(".focus").length)
536           {
537             //scrollen unterbinden
538             event.preventDefault();
539             trigger_class_split = $(".focus").attr("class").split("_");
540             // nächste Zeile ermitteln
541             row_number_next = parseInt(trigger_class_split[3]) + 1;
542
543             // hat gerade die letzte Zeile den Fokus?
544             if(!$(".card_" + trigger_class_split[1] + "_row_" + row_number_next).length)
545             {
546                 $("#card_" + trigger_class_split[1] + "_nextPage").click();
547             }
548             else
549             {
550                 focus_row(trigger_class_split[1], row_number_next);
551             }
552           }
553         break;
554         // Pfeiltaste hoch
555         case 38:
556           // hat gerade eine Zeile den Fokus?
557           if($(".focus").length)
558           {
559             trigger_class_split = $(".focus").attr("class").split("_");
560             trigger_row_number = parseInt(trigger_class_split[3]);
561
562             if(trigger_row_number>0)
563             {
564               row_number_previous = trigger_row_number - 1;
565               focus_row(trigger_class_split[1], row_number_previous);
566             }
567             // die erste Zeile hat den Fokus; es wird nicht die erste Seite ange-
zeigt
568             else if($("#card_" + trigger_class_split[1] + "_prevPage").length)
569             {
570               $("#card_" + trigger_class_split[1] + "_prevPage").click();
571             }
572             // die erste Zeile hat den Fokus; es wird die erste Seite angezeigt
573             else
574             {
575               // Suchfeld bekommt den Focus
576               $("#suchfeldliste_" + trigger_class_split[1]).focus();
577             }
578           }
579         break;
580       }
581   });
582 });
```

Anhang F - Quelltexte (JavaScript/jQuery) für das Ziehen und Loslassen von Cards

```
$(document).ready(function()
{
  const
    CELL_EDGELENGTH__PX = 100,
    CELL_BORDER__PX = 10,
    CELL_EDGELENGTH_AND_BORDER____PX = 110;

  var
    div_drag__id ,
    div_drag__id_split ,
    cell_origin__row ,
    cell_origin__column ,
    cell_origin__offset ,
    card_width ,
    card_height ,
    row_marker__origin ,
    cardcell_grabbed__row ,
    cardcell_grabbed__column ,
    raster_width,
    raster_height,
    card_id,
    cardfits = false;

    var div_raster__id_split = $(".raster").attr("id").split("_");
    raster_width = div_raster__id_split[0];
    raster_height = div_raster__id_split[1];

    $(".anchor").each(function()
    {
      var div_drag__atAnchor___id_split =
$(this).children(":first").attr("id").split("_");
      var cell_anchor_row = div_drag__atAnchor___id_split[1];
      var cell_anchor_column = div_drag__atAnchor___id_split[2];
      var card_atAnchor__width  = div_drag__atAnchor___id_split[3];
      var card_atAnchor__height = div_drag__atAnchor___id_split[4];
      var card_atAnchor__id = div_drag__atAnchor___id_split[5];
      cells_reserving(cell_anchor_row, cell_anchor_column, card_atAnchor__width,
card_atAnchor__height, card_atAnchor__id);
    });

    //cell_checking(1, 1, 2, 2, 0, 2);

    $.fn.dummy_writeText = function()
    {
      $(this).append("Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.");
      //$(document).cells_and_cards___positioning();
    }

     function cell_checking(cell_kickoff__row, cell_kickoff__column,
cell_limes__row, cell_limes__column, cell_row, cell_column)
    {
      // Zelle als Dropvorschlag markieren
      $("div[id^='drop_" + cell_row + "_" + cell_column + "_'").addClass("drop_suggestion");

      // Zelle besetzt?
      if($("div[id^='drop_" + cell_row + "_" + cell_column + "_']").is(".reserved"))
      {
        var div_drop__id_split = $("div[id^='drop_" + cell_row + "_" + cell_column +
"_']").attr("id").split("_");
        // erste Zellenspalte der Card?
        if(cell_kickoff__column == cell_column)
        {
          var card_expansion__left_int = parseInt(div_drop__id_split[3]);
          // hat die Zelle eine Card, welche sich nach links ausbreitet?
```

```
61            if(card_expansion__left_int > 0)
62            {
63              return false;
64            }
65          }
66          // erste Zellenreihe der Card?
67          if(cell_kickoff__row == cell_row)
68          {
69            var card_expansion__top_int = parseInt(div_drop__id_split[5]);
70            // hat die Zelle eine Card, welche sich nach oben ausbreitet?
71            if(card_expansion__top_int > 0)
72            {
73              return false;
74            }
75          }
76
77          // letzte Reihe der Card?
78          if(cell_limes__row == cell_row)
79          {
80            var card_expansion__under_int = parseInt(div_drop__id_split[6]);
81            if(card_expansion__under_int > 0)
82            {
83              return false;
84            }
85          }
86
87          // letzte Spalte der Card?
88          if(cell_limes__column == cell_column)
89          {
90             var card_expansion__right_int = parseInt(div_drop__id_split[4]);
91             if(card_expansion__right_int > 0)
92             {
93               return false;
94             }
95          }
96        }
97        // Ende der des Dropbereichs erreicht?
98        if(cell_row == cell_limes__row && cell_column == cell_limes__column)
99        {
100         return true;
101       }
102       else
103       {
104         if(cell_column==cell_limes__column)
105         {
106           cell_row = parseInt(cell_row) + 1;
107           cell_column = cell_kickoff__column;
108         }
109         else
110         {
111             cell_column = parseInt(cell_column) + 1;
112         }
113         return cell_checking(cell_kickoff__row, cell_kickoff__column,
cell_limes__row, cell_limes__column, cell_row, cell_column);
114       }
115     }
116
117     function cells_reserving(cell_kickoff__row, cell_kickoff__column, card_width,
card_height, card_id)
118     {
119       var cell_limes__row     =  parseInt(cell_kickoff__row) + par-
seInt(card_height);
120       var cell_limes__column  =  parseInt(cell_kickoff__column) + par-
seInt(card_width);
121
122       for (var cell_row = parseInt(cell_kickoff__row); cell_row < cell_limes__row;
cell_row++)
123         for (var cell_column = parseInt(cell_kickoff__column); cell_column <
cell_limes__column; cell_column++)
124         {
125           // Ausbreitung der Card ermitteln - ausgehend von der aktuellen Zelle
126           var expansion_left = cell_column - parseInt(cell_kickoff__column);
127           var expansion_right = cell_limes__column - cell_column - 1;
128           var expansion_top = cell_row - parseInt(cell_kickoff__row);
129           var expansion_under = cell_limes__row - cell_row - 1;
130
131           $("#drop_" + cell_row + "_" + cell_column).addClass("card_" + card_id);
```

```
132             $("#drop_" + cell_row + "_" + cell_column).addClass("reserved");
133
134             // Ausbreitung in der Zellen-Id speichern
135             $("#drop_" + cell_row + "_" + cell_column).attr("id", "drop_" + cell_row + "_" +
cell_column + "_" + expansion_left + "_" + expansion_right + "_" + expansion_top + "_" +
expansion_under);
136             //console.log("div[id^='drop_" + cell_row + "_" + cell_column +
"_']).addClass(card_" + card_id + ")");
137           }
138       }
139
140       function cards_replacing(cell_source__kickoff___row,
cell_source__kickoff___column, cell_source__limes___row, cell_source__limes___column,
cell_target__kickoff___row, cell_target__kickoff___column, cell_progress__row,
cell_progress__column)
141       {
142         var cell_source__row = cell_source__kickoff___row + cell_progress__row;
143         var cell_source__column = cell_source__kickoff___column +
cell_progress__column;
144
145         var cell_target__row = parseInt(cell_target__kickoff___row) + par-
seInt(cell_progress__row);
146         var cell_target__column = parseInt(cell_target__kickoff___column) + par-
seInt(cell_progress__column);
147
148
149         // gefundene Zelle besetzt?
150         if($("div[id^='drop_" + cell_source__row + "_" + cell_source__column +
"_']").is(".reserved"))
151         {
152           // Zellen-id wieder so verändern, dass nicht mehr die Cardausbreitung mit
beinhaltet ist
153           $("div[id^='drop_" + cell_source__row + "_" + cell_source__column + "_']").attr("id",
"drop_" + cell_source__row + "_" + cell_source__column);
154
155           // obere Linke Ecke der Zelle? = Anfang der Karte?
156           if($("#drop_" + cell_source__row + "_" + cell_source__column).is(".anchor"))
157           {
158             var div_source__id = $("div[id^='drag_" + cell_source__row + "_" +
cell_source__column + "_']").attr("id");
159             var div_source__id_split = div_source__id.split("_");
160             var source_row__marker = parseInt(cell_source__row) + 1;
161             var card_width = div_source__id_split[3];
162             var card_height = div_source__id_split[4];
163             var card_id = div_source__id_split[5];
164             var div_target__offset = cell_target__row + "_" + cell_target__column;
165             var div_target__id = "drag_" + div_target__offset + "_" + card_width + "_" +
card_height + "_" + card_id;
166             var target_row__marker = parseInt(cell_target__row) + 1;
167
168             // Rowmarker des Sourcedivs aktualisieren - neue Zeile ist natürlich die
Targetzelle
169             $("#" + div_source__id + ">.rowMarker_" + source_row__marker).attr("class",
"rowMarker_" + target_row__marker);
170             // Belegten Bereich an der Source freiräumen
171             $(".card_" + card_id).attr("class", "droppable ui-droppable");
172             // Sourcediv ans die Zielzelle setzen
173             $("#drop_" + div_target__offset).append($("#" + div_source__id));
174             console.log("$(#drop_" + div_target__offset + ").append($(#" + div_source__id +
"));");
175             // Neue Id für das verschobene Div
176             $("#" + div_source__id).attr("id", div_target__id);
177             // Zelle als Cardanfang markieren
178             $("#drop_" + div_target__offset).addClass("anchor");
179             // Zellen reservieren
180             //console.log("cells_reserving(cell_target__row = " + cell_target__row +
",cell_target__column = " + cell_target__column + ", card_width =" + card_width + ",
card_height =" + card_height + ", card_id =" + card_id);
181             cells_reserving(cell_target__row, cell_target__column, card_width,
card_height, card_id);
182
183           }
184       }
```

```
185       if(cell_source__row == cell_source__limes___row && cell_source__column ==
cell_source__limes___column)
186       {
187         console.log("cards_replacing() : : ready");
188       }
189       else
190       {
191         if(cell_source__column == cell_source__limes___column)
192         {
193           cell_progress__row = cell_progress__row + 1;
194           cell_progress__column = 0;
195         }
196         else
197         {
198           cell_progress__column = cell_progress__column + 1;
199         }
200         cards_replacing(cell_source__kickoff___row, cell_source__kickoff___column,
cell_source__limes___row, cell_source__limes___column, cell_target__kickoff___row,
cell_target__kickoff___column, cell_progress__row, cell_progress__column);
201       }
202     }
203
204     $.fn.cells_and_cards___positioning = function()
205     {
206       // Abstand nach oben
207       var row_offset__top = 0;
208
209       for (var row_active = 1; row_active < raster_height; row_active++)
210       {
211         var row_active__height_max = CELL_EDGELENGTH__PX;
212
213         $(".rowmarker_" + row_active).each(function()
214         {
215           if(this.offsetTop > row_active__height_max)
216           {
217             row_active__height_max = this.offsetTop;
218           }
219         })
220
221         row_offset__top = row_offset__top + parseInt(row_active__height_max) +
CELL_BORDER__PX;
222
223         $("div[id^='drop_" + row_active + "_']").css({ "top" : row_offset__top });
224       }
225     }
226
227   $(".draggable").draggable(
228   {
229     cursor: 'move' ,
230     opacity: 0.75 ,
231     revert: true,
232
233     start: function(event, ui)
234     {
235       // Alle später benötigten Informationen über die Dragcard und deren Ursprungs-
zelle speichern
236       div_drag__id = $(this).attr("id");
237
238       div_drag__id_split = div_drag__id.split("_");
239       cell_origin__row = div_drag__id_split[1];
240       cell_origin__column = div_drag__id_split[2];
241       cell_origin__offset = cell_origin__row + "_" + cell_origin__column;
242       card_width = div_drag__id_split[3];
243       card_height = div_drag__id_split[4];
244       card_id = div_drag__id_split[5];
245       row_marker__origin = parseInt(cell_origin__row) + 1;
246
247       $(".card_" + card_id).removeClass("reserved");
248
249       // Breite der Dragcard > 1?
250       if(card_width>1)
251       {
252         // relativen Spaltenwert der angepackten Zelle ermitteln; die Zelle bezieht
sich auf die Card
253         var card_offset__left_int =  parseInt(cell_origin__column) *
CELL_EDGELENGTH_AND_BORDER____PX;
```

```
254         var card_width_int = parseInt(card_width);
255         var card_width__px_int = card_width * CELL_EDGELENGTH__PX + (card_width_int
- 1) * CELL_BORDER__PX;
256         var cursor_offset__relativeToCard___left = parseInt(event.pageX) -
card_offset__left_int;
257         var devisor = card_width__px_int / card_width_int;
258         cardcell_grabbed__column = parseInt(cursor_offset__relativeToCard___left /
devisor);
259       }
260       else
261       {
262         cardcell_grabbed__column = 0;
263       }
264
265       // Höhe der Dragcard > 1?
266       if(card_height>1)
267       {
268         // relativen Reihenwert der angepackten Zelle ermitteln; die Zelle bezieht
sich auf die Card
269         var card_offset__top_int = parseInt(cell_origin__row) *
CELL_EDGELENGTH_AND_BORDER____PX;
270         var card_height_int = parseInt(card_height);
271         var card_height__px_int = card_height * CELL_EDGELENGTH__PX +
(card_height_int - 1) * CELL_BORDER__PX;
272         var cursor_offset__relativeToCard__right = parseInt(event.pageY) -
card_offset__top_int;
273         var devisor = card_height__px_int / card_height_int;
274         cardcell_grabbed__row = parseInt(cursor_offset__relativeToCard__right / de-
visor);
275       }
276       else
277       {
278         cardcell_grabbed__row = 0;
279       }
280     },
281     stop: function()
282     {
283       $("#" + div_drag__id).removeClass("drop_possible");
284       $("#" + div_drag__id).removeClass("drop_impossible");
285      }
286   }
287   );
288
289   $(".droppable").droppable(
290   {
291     tolerance: 'pointer' ,
292
293     over: function(event, ui)
294     {
295       var div_hover__id_split = $(this).attr("id").split("_");
296       var cell_hover__row = div_hover__id_split[1];
297       var cell_hover__column = div_hover__id_split[2];
298       var cell_anchor_new___row_int = parseInt(cell_hover__row) - card-
cell_grabbed__row;
299       var cell_anchor_new___column_int = parseInt(cell_hover__column) - card-
cell_grabbed__column;
300       var cell_limes__row = cell_anchor_new___row_int + parseInt(card_height) - 1;
301       var cell_limes__column = cell_anchor_new___column_int + parseInt(card_width) -
1;
302
303       cardfits = cell_checking(cell_anchor_new___row_int,
cell_anchor_new___column_int, cell_limes__row, cell_limes__column,
cell_anchor_new___row_int, cell_anchor_new___column_int);
304       if(cardfits)
305       {
306         $("#" + div_drag__id).addClass("drop_possible");
307       }
308       else
309       {
310         $("#" + div_drag__id).addClass("drop_impossible");
311       }
312     } ,
313
314     out: function()
315     {
316       $("#" + div_drag__id).removeClass("drop_possible");
```

```
317         $("#" + div_drag__id).removeClass("drop_impossible");
318       } ,
319       drop: function(event, ui)
320       {
321         // darf das Dragdiv an der Stelle gedroppt werden?
322         if(cardfits)
323         {
324           var div_drop__id_split = $(this).attr("id").split("_");
325           var cell_source__kickoff___row = parseInt(div_drop__id_split[1]) - card-
cell_grabbed__row;
326           var cell_source__kickoff___column = parseInt(div_drop__id_split[2]) - card-
cell_grabbed__column;
327
328           // Anzahl der Zeilen, um die das Div verschoben wurde
329           var rows_shifted = cell_source__kickoff___row - parseInt(cell_origin__row);
330
331           var cell_offset = cell_source__kickoff___row + "_" +
cell_source__kickoff___column;
332           var cell_source__limes___row = cell_source__kickoff___row + par-
seInt(card_height) - 1;
333           var cell_source__limes___column = cell_source__kickoff___column + par-
seInt(card_width) - 1;
334
335           if(rows_shifted < 0)
336           {
337             dif = card_height - rows_shifted;
338           }
339           else
340           {
341             dif = parseInt(card_height) - rows_shifted;
342           }
343
344           $("#" + div_drag__id).removeClass("drop_possible");
345
346           // Herkunftszellen des Dragdivs wieder freiräumen
347           $(".card_" + card_id).each( function()
348           {
349             var div_this__id = $(this).attr("id");
350             var div_this__id_new = div_this__id.substring(0, 8);
351             $(this).attr("id", div_this__id_new);
352           }
353           );
354
355           $(".card_" + card_id).attr("class", "droppable ui-droppable");
356
357           cards_replacing(cell_source__kickoff___row, cell_source__kickoff___column,
cell_source__limes___row, cell_source__limes___column, cell_origin__row,
cell_origin__column, 0, 0);
358
359           $("#drop_" + cell_offset).append($(ui.draggable));
360
361           $("#drop_" + cell_offset).addClass("anchor");
362
363           $(ui.draggable).css({'position':'relative','top':'0','left':'0'});
364
365           cells_reserving(cell_source__kickoff___row, cell_source__kickoff___column,
card_width, card_height, card_id);
366
367           // Dragdiv.id unbenennen
368           $(ui.draggable).attr("id","drag_" + cell_source__kickoff___row + "_" +
cell_source__kickoff___column + "_"  + card_width + "_" + card_height + "_"  + card_id);
369
370         }
371       }
372     }
373     );
374 }
375 );
```

Anhang G - CSS-Formatierungen

```
/*
    Document   : new_design
    Created on : 06.08.2009, 16:59:29
    Author     : Clemens
    Description:
        Purpose of the stylesheet follows.
*/

/*
    Divs und Tables
    Positionierung, Dimensionierung, Hintergründe, Rahmen
*/
body
{

}

.td a { color: #000000; text-decoration: none; }
.td a:visited { color: #000000; text-decoration: none; }
.td a:hover { color: #000000; text-decoration: none; }
.td a:focus { color:#000000; text-decoration: none; }



div.space24
{
    height: 24px;
    line-height: 24px;
}

div.space6
{
    height: 6px;
    line-height: 12px;
}

div#headbar_left
{
    margin: 0px;
    margin-top: 22px;
    padding: 0px;
    width: 13px;
    height: 60px;
    background: url(../images/bg_headbar_left.jpg) top left no-repeat;
    float: left;
}

div#headbar_body
{
    margin: 0px;
    margin-top: 22px;
    padding: 0px;
    height: 60px;
    background: url(../images/bg_headbar.gif) top left repeat-x;
    float:left;
    line-height: 55px;
}

div#headbar_right
{
    margin: 0px;
    margin-top: 22px;
    padding: 0px;
    width: 13px;
    height: 60px;
    background: url(../images/bg_headbar_right.jpg) top right repeat-y;
    float: left;
}

div#headbar_logo
{
    margin: 0px;
    padding: 0px;
```

```
74      width: 240px;
75      height: 60px;
76      background: url(../images/logo.jpg) top left no-repeat;
77      float: left;
78  }
79
80  div#headbar_customer_logo
81  {
82      margin: 0px;
83      padding: 0px;
84      float: left;
85  }
86
87  div#headbar_processes
88  {
89      margin: 0px;
90      padding: 0px;
91      float:left;
92      line-height: 55px;
93  }
94
95  div.card_border_left
96  {
97      margin: 0px;
98      padding: 0px;
99      background: url(../images/bg_card_border_left.gif) top left repeat-y;
100     background-color: #494949;
101 }
102
103 div.card_border_right
104 {
105     margin: 0px;
106     padding: 0px;
107     background: url(../images/bg_card_border_right.gif) top right repeat-y;
108 }
109
110 div.card_border_top_left
111 {
112     margin: 0px;
113     padding: 0px;
114     width: 11px;
115     height: 9px;
116     background: url(../images/bg_card_border_top_left.gif) top left no-repeat;
117     float: left;
118 }
119
120 div.card_body
121 {
122     margin: 0px;
123     padding: 0px;
124     min-height:337px;
125     height:auto !important;  /* für moderne Browser */
126     height:337px;  /*für den IE 6 */
127     background: url(../images/bg_card_border_top.gif) top left repeat-x;
128     float: left;
129     color: #ffffff;
130 }
131
132 div.card_body table
133 {
134   color: #000000;
135 }
136
137 div.card_border_top_right
138 {
139     margin: 0px;
140     padding: 0px;
141     width: 11px;
142     height: 9px;
143     background: url(../images/bg_card_border_top_right.gif) top left no-repeat;
144     float: left;
145 }
146
147 .card_headbar_left
148 {
149     margin: 0px;
150     padding: 0px;
```

```
151     width: 6px;
152     height: 42px;
153     background: url(../images/bg_card_headbar_left.jpg) top left no-repeat;
154     float: left;
155 }
156 
157 div.card_headbar
158 {
159     margin: 0px;
160     padding: 0px;
161     height: 42px;
162     background: url(../images/bg_card_headbar.gif) top left repeat-x;
163     overflow: hidden;
164     float: left;
165 }
166 
167 div.card_headbar_right
168 {
169     margin: 0px;
170     padding: 0px;
171     width: 6px;
172     height: 42px;
173     background: url(../images/bg_card_headbar_right.jpg) top left no-repeat;
174     float: left;
175 }
176 
177 div.card_border_bottom_left
178 {
179     margin: 0px;
180     padding: 0px;
181     width: 11px;
182     height: 12px;
183     background: url(../images/bg_card_border_bottom_left.gif) top left no-repeat;
184     float: left;
185 }
186 
187 div.card_border_bottom
188 {
189     margin: 0px;
190     padding: 0px;
191     height: 12px;
192     background: url(../images/bg_card_border_bottom.gif) top left repeat-x;
193     float: left;
194 }
195 
196 div.card_border_bottom_right
197 {
198     margin: 0px;
199     padding: 0px;
200     width: 11px;
201     height: 12px;
202     background: url(../images/bg_card_border_bottom_right.gif) top left no-repeat;
203     float:left;
204 }
205 
206 div.cleaner
207 {
208     margin: 0px;
209     padding: 0px;
210     height: 0px;
211     line-height: 1px;
212     font-size: 1px;
213     clear: both;
214 }
215 
216 div.card
217 {
218     min-height:375px;
219     height:auto !important;  /* für moderne Browser */
220     height:375px;  /*für den IE 6 */
221 }
222 
223 div.cardrow
224 {
225     width: 2250px;
226     background: #ffffff;
227 }
```

```
228
229 div.data
230 {
231     margin: auto;
232     padding: 0px;
233     background: #ffffff;
234 }
235
236 div.th_left
237 {
238     margin: 0px;
239     padding: 0px;
240     width: 7px;
241     height: 29px;
242     background: url(../images/bg_table_border_top_left.gif) top left no-repeat;
243     float:left;
244 }
245
246 div.th_wp2_lever
247 {
248     margin: 0px;
249     padding: 0px;
250     width: 12px;
251     height: 29px;
252     line-height: 29px;
253     background: url(../images/bg_table_border_top.gif) top left repeat-x;
254     overflow: hidden;
255     float: left;
256 }
257
258 div.tr_top
259 {
260   height: 29px;
261   background: url(../images/bg_table_border_top.gif) top left repeat-x;
262 }
263
264 div.th
265 {
266     margin: 0px;
267     padding: 0px;
268     height: 29px;
269     line-height: 29px;
270     overflow: hidden;
271     float: left;
272 }
273
274 div.th_hyphen
275 {
276     margin: 0px;
277     padding: 0px;
278     height: 29px;
279     display: block;
280     background: url(../images/bg_table_th_hyphen.gif) top center no-repeat;
281     font-size: 1px;
282     float: left;
283 }
284
285 div.th_working_icons
286 {
287     margin: 0px;
288     padding: 0px;
289     width: 31px;
290     height: 29px;
291     display: block;
292     background: url(../images/bg_table_border_top.gif) top left repeat-x;
293     text-align: right;
294     float: left;
295 }
296
297 div.th_right
298 {
299     margin: 0px;
300     padding: 0px;
301     width: 7px;
302     height: 29px;
303     display: block;
304     background: url(../images/bg_table_border_top_right.gif) top left no-repeat;
```

```
305     float: right;
306 }
307
308 div.td_left
309 {
310     margin: 0px;
311     padding: 0px;
312     width: 7px;
313     display: block;
314     background: url(../images/bg_table_border_left.gif) top left repeat-y;
315     float:left;
316 }
317
318 div.wp2_lever
319 {
320     margin: 0px;
321     padding: 0px;
322     font-size: 1px;
323     width: 12px;
324 }
325
326 div.wp2_lever img
327 {
328   margin: 0px;
329   padding: 0px;
330   vertical-align: middle;
331 }
332
333 div.td
334 {
335   display: block;
336   overflow: hidden;
337   float: left;
338   color: #000000;
339 }
340
341 div.data_boolean
342 {
343   position: relative;
344   display: block;
345 }
346
347 div.td input[type=checkbox]
348 {
349   z-index: 9999;
350 }
351
352 div.td_hyphen
353 {
354     margin: 0px;
355     padding: 0px;
356     display: block;
357     background: url(../images/bg_table_td_hyphen.gif) top center repeat-y;
358     font-size: 1px;
359     float: left;
360 }
361
362 div.working_icons
363 {
364     margin: 0px;
365     padding: 0px;
366     width: 31px;
367     text-align: center;
368     font-size: 1px;
369 }
370
371 div.working_icons img
372 {
373   margin: 0px;
374   margin-left: 3px;
375   padding: 0px;
376   vertical-align: middle;
377 }
378
379 div.working_icons input
380 {
381   margin: 0px;
```

```
382   margin-left: 3px;
383   padding: 0px;
384   vertical-align: middle;
385 }
386
387
388 div.td_right
389 {
390     margin: 0px;
391     padding: 0px;
392     width: 7px;
393     display: block;
394     background: url(../images/bg_table_border_right.gif) top left repeat-y;
395     float: right;
396 }
397
398 div.tr_bottom
399 {
400     height: 11px;
401     background: url(../images/bg_table_border_bottom.gif) top left repeat-x;
402 }
403
404
405 div.td_bottom_left
406 {
407     margin: 0px;
408     padding: 0px;
409     width: 7px;
410     height: 11px;
411     background: url(../images/bg_table_border_bottom_left.gif) top left no-repeat;
412     display: block;
413     float: left;
414 }
415
416 div.td_bottom_wp2_lever
417 {
418   margin: 0px;
419   padding: 0px;
420   width: 12px;
421   height: 11px;
422   background: url(../images/bg_table_border_bottom.gif) top left repeat-x;
423   display: block;
424   float: left;
425 }
426
427 div.td_bottom_wp2_lever img
428 {
429   margin: 0px;
430   padding: 0px;
431   width: 12px;
432   height: 11px;
433   background: url(../images/bg_table_border_bottom.gif) top left repeat-x;
434   display: block;
435   float: left;
436 }
437
438 div.td_bottom
439 {
440     margin: 0px;
441     padding: 0px;
442     height: 11px;
443     display: block;
444     float: left;
445 }
446
447 div.td_bottom_hyphen
448 {
449     margin: 0px;
450     padding: 0px;
451     height: 11px;
452     background: url(../images/bg_table_td_hyphen_bottom.gif) top center no-repeat;
453     display: block;
454     font-size: 1px;
455     float: left;
456 }
457
458 div.td_bottom_working_icons
```

```
{
    margin: 0px;
    padding: 0px;
    height: 11px;
    background: url(../images/bg_table_border_bottom.gif) top left repeat-x;
    display: block;
    width: 31px;
    float: left;
}

div.td_bottom_right
{
    margin: 0px;
    padding: 0px;
    width: 7px;
    height: 11px;
    background: url(../images/bg_table_border_bottom_right.gif) top left no-repeat;
    display: block;
    float: right;
}


div.edit
{
    margin: 0px;
    padding: 0px;
    width: 100%;
    border-top: 1px solid #000000;
    border-bottom: 1px solid #000000;
}

div.edit_forms
{
    margin: 0px;
    padding: 0px;
    float: left;
    height: 20px;
    line-height: 20px;
}

div.edit_icons
{
    margin: 0px;
    padding: 0px;
    width: 50px;
    height: 20px;
    background: #ffffff;
    float: right;
    line-height: 20px;
}

div.card_neck
{
    margin: auto;
    height: 28px;
}

div.card_neck__left
{
    margin: 0px;
    padding: 0px;
    height: 28px;
    float: left;
}

div.card_neck__right
{
    margin: 0px;
    padding: 0px;
    height: 28px;
    float: right;
}

div.card_paginator
{
    padding: 0px;
    margin: auto;
```

```
536     height: 24px;
537 }
538
539 div.paging_pageOfPages
540 {
541     margin: 0px;
542     padding: 0px;
543     height: 24px;
544     float: left;
545 }
546
547 div.paging_buttons
548 {
549     margin: 0px;
550     padding: 0px;
551     height: 24px;
552     float: right;
553     line-height: 1px;
554 }
555
556 /*  STYLES
557     - Farben, Schriften, Formelemente
558 */
559
560 a.headbar
561 {
562     color: #fefefe;
563     font: 13px  Arial, Verdana, Trebuchet, sans-serif;
564     font-weight: bold;
565 }
566
567 a.headbar_activate
568 {
569     color: #FF7D74;
570     font: 18px  Arial, Verdana, Trebuchet, sans-serif;
571     font-weight: bold;
572 }
573
574
575 div.card_headbar
576 {
577     color: #e6f7ff;
578     font: 13px  Arial, Verdana, Trebuchet, sans-serif;
579     font-weight: bold;
580     text-align: center;
581     line-height: 36px;
582 }
583
584 th.border_top
585 {
586     color: #272727;
587     font: 11px  Arial, Verdana, Trebuchet, sans-serif;
588     line-height: 29px;
589 }
590
591 div.edit_icons input
592 {
593     vertical-align: middle;
594 }
595
596 div.data_integer,
597 div.data_decimal,
598 div.data_float,
599 div.head_integer,
600 div.head_decimal,
601 div.head_float
602 {
603     text-align:right;
604 }
605
606 input.search_button
607 {
608     margin: 0px;
609     padding: 0px;
610     width: 23px;
611     height: 23px;
612     border: none;
```

```
613     background: url(../images/bg_search_button.gif) top left no-repeat;
614     line-height: 23px;
615 }
616
617 p.card_info
618 {
619     font-size: 9px;
620     color: #b7b7b7;
621     margin: 0px;
622     padding: 0px
623 }
624
625 .card_icon
626 {
627     margin: 0px;
628     padding: 0px;
629     width: 24px;
630     height: 24px;
631     display: inline;
632     border: none;
633     vertical-align:top;
634 }
635
636 input[type=text]
637 {
638     margin: 0px;
639     padding: 0px;
640     padding-top: 1px;
641     padding-left: 1px;
642     height: 15px;
643     background: url(images/bg_form_text.gif) top left repeat-x;
644     background-color: #ffffff;
645     border: 1px solid #CFCFCF;
646     display: inline;
647     vertical-align:top;
648     line-height: 15px;
649 }
650
651
652 input[type=text].worked, input[type=password].worked, input[type=checkbox].worked,
input[type=radiobutton].worked, textarea.worked, select.worked
653 {
654     border: 1px solid #ffd700;
655 }
656
657 p input[type=text].worked, p input[type=password].worked, p in-
put[type=checkbox].worked, p input[type=radiobutton].worked, p textarea.worked, p se-
lect.worked
658 {
659   border: 1px solid #ffd700;
660 }
661
662 input[type=text].has_errors
663 {
664   border: 1px solid #ff0000;
665 }
666
667 p input[type=text].has_errors
668 {
669   border: 1px solid #ff0000;
670 }
671
672 p textarea
673 {
674     margin: 0px;
675     padding: 0px;
676     padding-top: 1px;
677     padding-left: 1px;
678     background: url(images/bg_form_text.gif) top left repeat-x;
679     background-color: #ffffff;
680     border: 1px solid #002b42;
681     display: inline;
682     vertical-align:top;
683     font-size: 11px;
684     width: 285px;
685 }
686
```

```
687 input.search_text
688 {
689     margin: 0px;
690     padding: 0px;
691     padding-top: 4px;
692     padding-left: 2px;
693     width: 175px;
694     height: 23px;
695     border: none;
696     background: url(../images/bg_search_text.gif) top left no-repeat;
697     line-height: 23px;
698 }
699
700 label
701 {
702     color: #5fa4c7;
703 }
704
705 .pagination a
706 {
707     color: #f0f0f0;
708 }
709
710 table.adress
711 {
712     margin: auto;
713     margin-top: 5px;
714     padding: 0px;
715 }
716
717 div.suchfelddiv p,
718 div.suchfelddiv input.suchergebnis_klartext,
719 div.suchfelddiv div.suchfeldrahmen a.suche_pseudo_input
720 {
721   display: block;
722   float: left;
723   text-decoration: none;
724   width: 284px;
725   height: 17px;
726   font-size: 10px;
727   padding: 1px 0 1px 2px;
728   border: 1px solid #000000;
729   background: url(images/bg_form_text.gif) top left repeat-x;
730   background-color: #ffffff;
731   color: #000000;
732   line-height: 17px;
733   overflow: hidden;
734 }
735
736 input[type=text].page_text
737 {
738     margin: 0px;
739     padding: 0px;
740     width: 30px;
741     height: 18px;
742     border: 1px solid #434343;
743     background: #545454;
744     line-height: 18px;
745     font: 13px  Arial, Verdana, Trebuchet, sans-serif;
746     font-weight:bold;
747     color: #ffffff;
748     text-align: center;
749     vertical-align: middle;
750 }
751
752 p input[type=text], p select
753 {
754     margin: 0px;
755     padding: 0px;
756     padding-top: 1px;
757     padding-left: 1px;
758     height: 15px;
759     background: url(images/bg_form_text.gif) top left repeat-x;
760     background-color: #ffffff;
761     border: 1px solid #002b42;
762     display: inline;
763     vertical-align:top;
```

```
764     line-height: 15px;
765     width: 285px;
766     color: #000000;
767 }
768
769 div.paging_pageOfPages
770 {
771     font: 13px  Arial, Verdana, Trebuchet, sans-serif;
772     font-weight:bold;
773     color: #ffffff;
774     line-height: 22px;
775 }
776
777 div#container div#cardcontainer h2
778 {
779     font: 13px  Verdana, Arial, Trebuchet, sans-serif;
780     font-weight: bold;
781     text-indent: 6px;
782     height: 16px;
783     line-height: 16px;
784 }
785
786 div.focus
787 {
788     background-color: #f0f0d5;
789 }
790
791 div.highlight
792 {
793   background-color: #faada4;
794 }
795
796 ul#attachments
797 {
798   margin: 0px;
799   padding: 0px;
800   list-style-type:none;
801 }
802
803 ul#attachments li
804 {
805   margin: 0px;
806   padding: 0px;
807   text-indent: 6px;
808   height: 18px;
809   line-height: 18px;
810 }
811
812 ul#attachments li a
813 {
814   color: #ffffff;
815 }
816
817 ul.validation_errors
818 {
819   display: none;
820 }
821
822 ul.validation_errors li
823 {
824   color: #FF7D74;
825 }
826
827 div.wp1_column_relieved
828 {
829   margin: 0px;
830   padding: 0px;
831   z-index: 9998;
832   position: absolute;
833   background: #ffffff;
834   height: 20px;
835 }
836
837 div#wp2
838 {
839   overflow: hidden;
840 }
```

```
841
842 div#wp2_ground
843 {
844   float: left;
845 }
846
847 div.wp2_field
848 {
849   margin: 0px;
850   padding: 0px;
851   position: relative;
852   width: 1px;
853   height: 1px;
854 }
855
856 div.wp2_label
857 {
858   margin: 0px;
859   padding: 0px;
860   height: 16px;
861   line-height: 16px;
862   color: #000000;
863   overflow: visible;
864 }
865
866 div#footer {
867
868   margin: 0px;
869   margin-top: 10px;
870   display: none;
871   width: 99%;
872   height: 18px;
873   background-color: #eee;
874   font-size: 9px;
875   text-align: left;
876   line-height: 18px;
877 }
878
879 a.extended
880 {
881   width: 100%;
882   height: 100%;
883   display: block;
884 }
885 /*  */
886
887 #tooltip
888 {
889   margin: 0px;
890   padding: 0px;
891   padding-left: 2px;
892   padding-right: 2px;
893   background: #fdffc2;
894   position:absolute;
895   border:1px solid #4b4b4b;
896   color:#000000;
897   display:none;
898   z-index: 9999;
899   line-height:18px;
900 }
901
902 #spinner {
903   position: absolute;
904   width: 32px;
905   height: 32px;
906   top: 50%;
907   left: 50%;
908   margin-top: -16px;
909   margin-left: -16px;
910   z-index: 9999;
911 }
```

Anhang H - Fragebogen zur Benutzeroberfläche von onrooby

onrooby GmbH

David-Gilly-Str. 1

D-14469 Potsdam

Ansprechpartner: Clemens Teichmann (Tel.: 0331 505 44 21 / E-Mail: clemens.teichmann@onrooby.com)

Fragebogen zur Benutzeroberfläche von onrooby

Dieser Fragebogen ist zur Qualitätssicherung der Benutzeroberfläche von onrooby. Ihre Antworten werden anonym behandelt und statistisch ausgewertet. Das Ausfüllen der 17 Fragen dauert ca. 10 Minuten.

1 Design und Aufbau der Oberfläche

Wie bewerten Sie den Aufbau der Oberfläche?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○
sehr schlecht sehr gut

Wie gefällt Ihnen das Design?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○
sehr schlecht sehr gut

2 Cards

Wie bewerten Sie das Positionieren von Cards?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○
sehr schlecht sehr gut

Wie bewerten Sie das Dimensionieren von Cards?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○
sehr schlecht sehr gut

3 Listcards

Wie bewerten Sie den Aufbau von Listcards?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○
sehr schlecht sehr gut

Wie häufig bearbeiten Sie einen Datensatz in einer Zeile?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○
sehr schlecht sehr gut

Wie bewerten Sie das Bearbeiten eines Datensatzes in einer Zeile?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○
sehr schlecht sehr gut
sehr gut

Wie häufig benutzen Sie die aufklappbaren Formular-Felder in einer Liste?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

sehr schlecht sehr gut

Wie bewerten Sie das Arbeiten mit den aufklappbaren Formular-Feldern?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

sehr schlecht sehr gut

4 Detailcards

Wie bewerten Sie den Aufbau von Detailcards?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

sehr schlecht sehr gut

5 Nicht gespeicherte Daten

Wie gut werden nicht gespeicherte Daten gekennzeichnet?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

sehr schlecht sehr gut

Wie häufig gehen Ihnen nicht gespeicherte Daten ungewollt verloren?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

nie sehr oft

6 Visuelle Effekte

Wie bewerten Sie die visuellen Effekte?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

sehr schlecht sehr gut

Wie können Sie Card-übergreifende Beziehungen erkennen?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

sehr schlecht sehr gut

7 Workflow-Recorder

Wie häufig benutzen Sie den Workflow-Recorder?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

nie sehr oft

Wie gut lassen sich Workflows aufzeichnen?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

sehr schlecht sehr gut

Wie gut lässt sich mit aufgezeichneten Workflows arbeiten?

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

sehr schlecht sehr gut

8 Teilnehmer-spezifische Angaben

Mit welchem Browser arbeiten Sie?

_____

Mit welcher Auflösung arbeiten Sie?

_____

Wie viele Stunden haben Sie bisher schätzungsweise mit onrooby gearbeitet?

_____

9 Kommentare

.....

.....

.....

.....

.....

.....

Vielen Dank für Ihre Teilnahme!

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Hertel, Joachim: *Warenwirtschaftssysteme: Grundlagen und Konzepte*. - 1.Aufl. - Heilbronn: Physica, 1999

[2] Hussein, Morsy; Otto, Tanjs: *Ruby on Rails 2: Das Entwickler-Handbuch*. - 1.Aufl. - Bonn: Galileo Press, 2008

[3] Carl, Denny: *Praxiswissen Ruby on Rails*. - 1.Aufl. - Köln: O'Reilly, 2007

[4] *Ruby on Rails*. URL: <http://rubyonrails.org/>, verfügbar am 1.12.2009

[5] Heinemeier Hansson, David <david@loudthinking.com>: *Merb gets merged into Rails 3!*. URL: <http://weblog.rubyonrails.org/2008/12/23/merb-gets-merged-into-rails-3>, verfügbar am 1.12.2009

[6] Stewart, Bruce <bruce@oreilly.com>: *An interview with Yukihiro Matz Matsumoto, about Ruby, and his new book, Ruby in a Nutshell.* URL: <http://linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html>, verfügbar am 1.12.2009

[7] Venners, Bill: *The Philosophy of Ruby.* URL: <http://www.artima.com/intv/ruby.html>, verfügbar am 1.12.2009

[8] Redaktion SELFHTML <selfhtml81@selfhtml.org>: *Einführung in JavaScript und DOM*. URL: <http://de.selfhtml.org/javascript/intro.htm>, verfügbar am 1.12.2009

[9] Garrett, Jesse James: *Ajax: A New Approach to Web Applications*. URL: <http://www.adaptivepath.com/ideas/essays/archives/000385.php>, verfügbar am 1.12.2009

[10] Bucher, Jörg <bucher.joerg@easy-mind.de>: *NWA - Nutzwertanalyse als Entscheidungshilfe mit Beispielen.* URL: <http://community.easymind.info/page-76.htm>, verfügbar am 1.12.2009

[11] Redaktion SELFHTML <selfhtml81@selfhtml.org>: *Welche Unterschiede zwischen XHTML 1.0 und HTML 4.01 zu beachten sind*. URL: <http://de.selfhtml.org/html/xhtml/unterschiede.htm>, verfügbar am 1.12.2009

[12] Hoffmann, Manuela: *Modernes Webdesign*. - 1.Aufl. - Bonn: Galileo Press, 2008

[13] W3C - World Wide Web Consortium: *QA - What are the differences between png and gif*. URL: <http://www.w3.org/QA/Tips/png-gif>, verfügbar am 1.12.2009

[14] Paaßen, Urte <webmaster@araneae-online.net>: *Informationen über Barrierefreiheit einer Homepage im Internet und die Validierung der Seiten*. URL: <http://www.araneae-online.net/entwurf/barriere.html>, verfügbar am 1.12.2009

[15] Pratzner, Axel <kontakt@fragebogen.de>: *3.3 Aufbau des gesamten Fragebogens*.
URL: < http://www.fragebogen.de/aufbau-des-fragebogens.htm>, verfügbar am 1.12.2009

[16] Pratzner, Axel <kontakt@fragebogen.de>: *3.4 Grundlegender Aufbau des Hauptteils*.
URL: <http://fragebogen.de/grundlegender-aufbau-umfrage.htm>, verfügbar am 1.12.2009

Erklärung zur selbstständigen Anfertigung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Bearbeitungsort, Datum Unterschrift